ASSEMBLER PARA O MSX

JOSÉ EDUARDO M. DE CARVALHO

Assembler para o MSX

José Eduardo M de Carvalho

McGraw-Hill
São Paulo
Rua Tabapuã, 1.105, Itaim-Bibi
CEP 04533
(011) 881-8604 e (011) 881-8528

Rio de Janeiro • Lisboa • Porto • Bogotá • Buenos Aires • Guatemala • Madrid • México • New York • Panamá • San Juan • Santiago

Auckland • Hamburg • Kuala Lumpur • London • Milan • Montreal • New Delhi • Paris • Singapore • Sydney • Tokyo • Toronto

Assembler para o MSX

Editor : Milton Mira de Assumpção Filho
Coordenadora de Revisão : Daisy Pereira Daniel
Supervisor de produção: José Rodrigues

Dados de Catalogação na Publicação (CIP) Internacional
(Câmara Brasileira do Livro, SP, Brasil)

Carvalho, José Eduardo Maluf de.
C324a. Assembler para o MSX / José Eduardo Maluf de Carvalho. -- São Paulo : McGraw-Hill, 1987.

1. Assembler (Linguagem de programação para computadores) 2. MSX (Computadores) - Programação I. Título.

87-1746 CDD-001.6424
-001.64

Índices para catálogo sistemático:

1. Assembler: Linguagem de programação : Computadores : Processamento de dados 001.6424
2. MSX : Computadores : Processamento de dados 001.64

Existe uma pessoa especial a quem eu dedico este livro, pela imensa ajuda e incentivo que sempre me deu, inclusive montando este volume: VIVIAN, minha esposa.

AGRADECIMENTOS

São muitas as pessoas envolvidas na elaboração de um trabalho como este, seja na hora da criação, seja na hora da produção.

Meu primeiro agradecimento especial é a este grande companheiro, Milton Mira de Assumpção Filho, Editor responsável pela Editora McGraw Hill Ltda., pelo seu profundo senso profissional e pela sua grande confiabilidade e credibilidade em mim.

Outras pessoas, que indiretamente me incentivaram a chegar a este ponto, também devem ser citadas.
Desculpem-me as que foram esquecidas, não menos importantes.

Ricardo, Mário, Cesar, Paolo e Jan, meus ex-programadores da Tropic Informática que muito me ensinaram sobre este micro.

Grandes amigos na Tropic, como Toninho, Regina, Chico, Luís, Suzi, enfim, também merecem meus agradecimentos, pois só tinham palavras de incentivo para mim.

Meus fãs incondicionais, meu muito obrigado especial, pois devo muito a eles - minha mãe, meu pai, minha sogra e meu sogro.

Meu último agradecimento, o mais importante por sinal, vai para aquelas pessoas que participam mais diretamente da vida da gente, comemorando com a mesma alegria nossas vitórias e chorando junto as derrotas que sofremos - minha amada esposa **Vivian**, e meus adoráveis filhos **Felipe** e **Marina**.

A todos, **MUITO OBRIGADO MESMO!**

Maio de 1987

TUCA (O autor)

O AUTOR

Nos seus dois primeiros livros, *Basic para o TK 90X* e *Assembler para o TK 90X*, José Eduardo Maluf de Carvalho era descrito como um arquiteto, atuando em Planejamento urbano e Projetos e Construção Civil em seu escritório de Arquitetura, tratando a informática como um grande hobby, que lhe proporcionava horas de prazer. De repente surgiu a oportunidade de abrir uma micro empresa de informática, a Arquitron Informática Ltda. O seu hobby estava sendo profissionalizado.

Passado algum tempo, hoje a situação se inverteu completamente. Ele desligou-se totalmente dos seus laços com a Arquitetura, para dedicar-se exclusivamente à Tropic Informática Ltda., como Gerente de Desenvolvimento de Software, onde trabalhou cerca de dois anos intimamente ligado a micros da linha MSX.

No início estava relutante em abandonar os micros da linha Sinclair, pois até viajara para Londres, com o propósito de conhecer melhor aquela linha de microcomputadores.

Agora, não é preciso dizer mais nada: este livro está sendo redigido num microcomputador Hotbit, com o software "TASSWORD" e impresso numa GRAFIX 80 FT, depois de muito trabalho. Este episódio eu faço questão de contar:

Para imprimir este livro, primeiramente aluguei uma GRAFIX 80 T, velha e lenta. Após um certo trabalho, consegui ensiná-la a escrever corretamente o português, que este micro escreve. Resolvi então comprar uma impressora nova, a GRAFIX 80 FT, que realmente é uma ótima máquina. Pena que fosse meio "burrinha", pois não sabia escrever corretamente o português.

Não conseguia modificar o set de caracteres da sua EPROM original, pois não tinha os códigos dos caracteres acentuados, e aquele manual que vem junto com a máquina deixa muito a desejar. Consegui então com meu amigo Rodolpho, da Digital Design, uma EPROM "meio Abicomp" da Grafix, juntamente com os códigos dos seus caracteres. Finalmente, o texto foi acentuado, após dias e dias de quebra cabeças e adivinhações. Acho que os fabricantes deveriam entrar num acordo e lançar os sets Abicomp 1.1, 1.2, 1.2.05, 1.5 etc, pois agora este software está compatível somente com o Hotbit.

Veio então a profissionalização total da informática na vida do autor, que passou a dedicar poucas horas ao seu hobby predileto hoje, a Arquitetura.

Que prazer tem o autor ao utilizar uma máquina tão poderosa quanto este MSX, que apesar do seu microprocessador de 8 bits, não deixa nada a desejar aos poderosos micros "nacionais" de 16 bits.

Após todo esse convívio, juntamente com todo o conhecimento do autor sobre o microprocessador Z80, a interação autor/MSX foi tanta que na sua mesa de trabalho o microcomputador ZX Spectrum 128 da Sinclair está na lateral, cedendo seu lugar de honra ao MSX.

SUMÁRIO

APÊNDICES

INTRODUÇÃO

Valho-me aqui de uma adaptação da introdução do meu livro *Assembler para o TK 90X* .

Parabéns para você que deseja aprender a linguagem de máquina deste poderoso microprocessador de 8 bits, denominado Z80A.

Você provavelmente já deve ter dominado todo o potencial da linguagem Basic do seu micro, e, não satisfeito, deseja mais, deseja explorar mais as suas cores, o seu som e a sua velocidade de processamento através da linguagem de máquina.

Pois vá em frente! Não existe nenhuma linguagem de alto nível que explore todo o potencial da máquina. Lá, você trabalha com linhas de programa, comando a comando, que serão posteriormente executados, um a um, sequencialmente.

Aqui não - a estrutura da linguagem de máquina é completamente diferente. Evidente que a seqüência lógica de execução permanece, mas aqui manipulamos bytes diretamente armazenados na memória, através de endereços

preestabelecidos. E aqui está a primeira relação biunívoca da linguagem de máquina: a cada endereço corresponde um e somente um byte de 8 bits.

Você vai precisar de uma grande dose de paciência, deverá ser adivinho em algumas ocasiões, e principalmente deverá tomar muito cuidado com os valores numéricos que vai manipular. E exercitar muito.

Você já sabe que a única coisa que seu micro entende são sinais, respectivamente "com voltagem" e "sem voltagem", ou ainda "voltagem alta" e "voltagem baixa". No caso do Z8Ø, a voltagem alta equivale a + 5 volts, e a voltagem baixa equivale a Ø volts, que convencionou-se padronizar através dos dígitos Ø, para voltagem baixa, e 1 para voltagem alta., dando origem ao sistema binário, diretamente manipulado pelo micro.

Esses algarismos são mais conhecidos por bits, e o agrupamento de 8 desses bits dá origem a um byte, que é a menor unidade de armazenamento de memória de um micro, cujos valores vão de Ø a 255, no sistema decimal (representado daqui em diante por uma letra "d" após o número), totalizando 256 valores, ou, no sistema hexadecimal, de valores que iniciam em ØØØØ e terminam em FFFF (esse sistema numérico será representado daqui em diante somente pelo símbolo "H" antes do valor em questão - quando houver um valor numérico sem nenhuma letra após, significa que ela está expressa no sistema hexadecimal, que será o mais utilizado no decorrer deste livro).

Na linguagem Basic, chegamos a manipular bytes da memória, mas, veja a grande dificuldade da linguagem de máquina - aqui, manipularemos apenas um bit de um determinado byte, para ordenar algo ao micro. E, por esse caminho, você percebe que, se errar ou esquecer apenas 1 bit de 1 byte, põe a perder todo o trabalho de elaboração de uma rotina em linguagem de máquina, provocando um "crash" no sistema, ou seja, o não-retorno do micro para a linguagem residente. Nestes casos, não

se preocupe - o melhor a fazer é desligar e ligar o micro novamente.

Em qualquer linguagem de computação, você deve ter em mente, muito bem definida, a concepção geral do seu programa, sobre como ele vai funcionar, como vai responder a determinadas condições etc.

De posse dessa concepção, evidentemente que adequada à lógica do micro, você deve em seguida elaborar um fluxograma dele, ou seja, as suas etapas de execução.

Definidas estas etapas, com uma listagem das mais de 700 instruções em código de máquina, você passa então a escrever a sua rotina, utilizando as "mnemônicas", ou símbolos (nomes) das instruções, sempre levando em consideração os endereços, ou locações iniciais ou intermediárias da memória, que também podem ser rotulados, para não se perder em números mais tarde.

Nesta fase, você tem duas opções para continuar o trabalho:

1- Usar um programa ferramenta, denominado ASSEMBLER, ou montador, que permite que você digite a sua listagem ASSEMBLY, ou montada, a partir das mnemônicas, endereço por endereço, byte a byte, para que ele converta automaticamente estes seus símbolos em códigos, números binários que serão entendidos pelo micro para serem processados posteriormente.

Na realidade, o que se faz com as mnemônicas e valores hexadecimais é o que costumamos chamar de linguagem ASSEMBLER, ou mais costumeiramente linguagem hexadecimal. Nesta etapa, você está digitando, ou escrevendo, o PROGRAMA-FONTE, que após ser convertido para valores expressos no sistema numérico binário, pronto para ser executado, transformar-se-á no PROGRAMA-OBJETO.

2- Esta opção é praticamente continuação da anterior. De posse da listagem das mnemônicas do programa, você deve converter esses símbolos para os seus respectivos códigos no sistema hexadecimal.

Convertida a listagem para códigos hexadecimais, a última etapa desta opção é o armazenamento desses valores (que também podem estar expressos no sistema decimal, se você for armazená-los através da linguagem Basic) nos endereços da memória RAM que você determinar. Nesta etapa, nada lhe impede de usar um outro programa ferramenta, denominado EDITOR ASSEMBLER, que permite ao usuário digitar diretamente as mnemônicas, bem como seus códigos, expressos no sistema hexadecimal.

Como você deve ter notado, o trabalho de programação em código de máquina é muito exaustivo e sujeito a muitos erros, mas também muito compensador.

A prática desta linguagem é um fator muito importante para seu perfeito domínio, além do pleno conhecimento de todo o conjunto de instruções do microprocessador Z80A.

Praticando, e muito, você vai guardar os códigos, tanto em hexadecimal quanto em decimal, das instruções mais utilizadas na programação desta linguagem. E vai passar a ser conhecido como HEXAMEN!

De posse da sua rotina, ou o seu programa armazenado na memória RAM do seu micro, a etapa seguinte não é a sua execução - lembre-se de que é muito fácil errar - mas sim armazenar num periférico externo, tanto cassete como drive.

Agora que o programa já está salvo, você pode tentar executá-lo e saber, caso não aconteça o que você estava imaginando, onde está o erro, conferindo naquele papel rascunho onde você escreveu o programa, ou com o programa EDITOR-ASSEMBLER, que mostra a você, após os

devidos procedimentos (carregá-lo, para em seguida carregar o seu programa), a listagem do seu programa "DISASSEMBLADA", ou seja, uma listagem dos endereços, das mnemônicas e seus respectivos códigos de operação.

Corrija a rotina, da maneira que achar mais conveniente, seja em Basic, ou com o EDITOR ASSEMBLER, salve-a novamente e tente executá-la, repetindo o processo até atingir o objetivo final. Não pense que é fácil colocar vida infinita naquele jogo que você mais gosta, para saber o que acontece no final!

Quando atingir seu resultado parabenize-se, pois estará também programando em assembler, ou linguagem de máquina.

Este é o objetivo deste livro.

Ensiná-lo a programar em linguagem de máquina. Tentei abordar todos os aspectos necessários para a perfeita compreensão e domínio da linguagem, de uma maneira muito didática, considerando que você não entende nada do assunto, e que obrigatoriamente deve partir dos princípios mais rudimentares do processo (foi assim que eu comecei - sem curso algum).

Considerei apenas que você entendeu todos os capítulos dos manuais do seu micro.

A teoria deste livro é muito extensa - não tenha pressa, e não pule etapas, você deve conhecer a fundo como é o sistema do seu micro, bem como o funcionamento do micro processador dele. Tente assimilar tudo da melhor forma possível, para seguir em frente.

Começaremos estudando o que é este microprocessador Z80, com uma literatura técnica muito extensa e variada, mas quase nunca aplicada a um sistema de computador, como nesta obra. Em seguida, veremos a estrutura e a matemática na programação assembler, exercitando muito nas conversões de sistemas numéricos. Estudaremos,

também, as operações lógicas, para então entrarmos no extenso grupo de instruções do microprocessador Z80 .

Em seguida, vamos descrever a operação da PPI (*Programmable Peripheral Interface* ou *Interface programável de Periféricos*), do VDP (*Video Display Processor*– conforme o nome diz, o processador de vídeo) e do PSG (*Programmable Sound Generator* – processador de som).

Estes componentes são:

1- Microprocessador Z80A da Zilog.
2- PPI - Intel 8255
3- VDP - Texas 9128
4- PSG - AY 8910 - General Instrument

Os três chips (8255, 9128 e 8910) permitem o interfaceamento entre o Z80 e o hardware periférico do sistema MSX padrão. Todos eles ocupam posições preestabelecidas nas vias de endereçamento de entrada e saída do Z80. (I/O do Z80 bus).

Em seguida analisaremos a ROM do MSX, subdividida em duas partes. A primeira, denominada ROM BIOS (*Basic Input/Output System*), muito útil, e a segunda, o Interpretador Basic e o mapeamento da memória utilizado pelo sistema MSX padrão.

Finalmente, alguns exemplos de rotinas de programas em código de máquina, que utilizam as rotinas da ROM para economizar memória e ganhar velocidade de processamento, e os apêndices, necessários e úteis a qualquer programador de linguagem de máquina.

NOTA: Toda a terminologia utilizada neste livro, bem como a descrição do sistema MSX, foi baseada em literatura original, tanto da Microsoft como outras, inglesas e japonesas até!!! Portanto, já que existem algumas diferenças entre os micros MSX nacionais (para variar um pouco...), não estranhe se você ler alguma

coisa diferente do seu micro, principalmente se ele for um EXPERT da Gradiente, já que eu possuo somente um HOTBIT em minha casa (juntamente com os meus Sinclair e PC), e não houve tempo hábil, antes da execução deste livro, para que eu entrasse em contato com a Gradiente, a fim de descrever o EXPERT também (existem algumas diferenças entre o HOTBIT e o EXPERT, na ROM e na paginação da memória, principalmente).

Agora, mãos à obra!!!

CAPÍTULO 1 — FUNÇÃO USR E LINGUAGEM DE MÁQUINA

Para executar sub-rotinas em linguagem de máquina, a partir da linguagem Basic, utilize a função USR. Esta função chama o programa em linguagem de máquina que começa num endereço especificado numa declaração DEF USR. Você pode utilizar até 10 rotinas em código de máquina, simultaneamente. É possível ter-se mais de 10 rotinas sendo executadas simultaneamente, através da redefinição da função USR.

COMO DEFINIR A FUNÇÃO USR

Para chamar ou acessar uma rotina em linguagem de máquina, o micro precisa saber qual o endereço de início da rotina. Use a declaração DEF USR para associar o endereço à função USR.

Por exemplo: uma rotina em linguagem de máquina qualquer, que começa no endereço &HFØØØ.

DEF USRØ=&HFØØØ (Você pode abreviar o Ø de forma que DEF USR=&HFØØØ)

COMO EXECUTAR UMA ROTINA EM LINGUAGEM DE MAQUINA

A função USR pode ser utilizada da mesma maneira que uma função qualquer. Isto significa que elas podem estar em uma expressão, como a do exemplo a seguir:

```
M=USR(88)
T$="T/E"+USR9("TUCA")
PRINT USR7(Ø)
```

Os exemplos acima executam as sub-rotinas e retornam valores que dependem do que a rotina em questão faz. Os valores numéricos ou as "strings" (cadeia de caracteres) entre parênteses são parâmetros a serem utilizados pelos códigos de máquina. O modo de como você usa a função USR realmente depende da sua rotina em linguagem de máquina. Você pode ter um argumento nela, ou apenas um simples parâmetro; ela tanto pode retornar um simples parâmetro como também pode retornar nada. Se você deseja executar linguagem de máquina sem parâmetros de entrada, ou de saída, então utilize um nome, conforme exemplo a seguir:

```
PRÊMIO=USR(Ø)
```

Onde PRÊMIO é uma variável e (Ø) é o parâmetro da variável.

COMO PASSAR PARAMETROS DA LINGUAGEM BASIC PARA A LINGUAGEM DE MAQUINA

Você pode passar qualquer tipo de parâmetro para rotinas em linguagem de máquina, a partir do Basic, usando

USR<NÚMERO> (<PARÂMETRO>)

O parâmetro deve estar entre parênteses. Quando o MSX executa a função, ele checa o tipo de argumento usado, para saber se é um inteiro, um valor de precisão simples ou um valor de precisão dupla, ou uma string. Se houver um parâmetro, a linguagem de máquina precisa saber de que tipo ele é, e onde está armazenado na memória. O Basic do MSX ordena os tipos de parâmetros antes que uma rotina em linguagem de máquina faça uso deles na sua execução.

Para descobrir que tipo de parâmetro está sendo passado, veja os endereços &HF663:

```
&hF663=2=00000010=parâmetro inteiro
&hF663=4=00000100=parâmetro para
                  precisão simples
&hF663=8=00001000=parâmetro para
                  precisão dupla
&hF663=3=00000011=parâmetro string
```

Locações dos parâmetros a serem utilizados:

```
Inteiro(&HF663=2)
        &HF7F8= byte menos significativo
        &HF7F9= byte mais  significativo

Número de precisão simples (&HF663=4)
        &HF7F6=
        &HF7F7=    dados armazenados em BCD
        &HF7F8=    a partir de &HF7F6
        &HF7F9=

Número de precisão dupla   (&HF663=8)
        &HF7F6=
        &HF7F7=
        &HF7F8=
        &HF7F9=    dados armazenados em BCD
        &HF7FA=    a partir de &HF7F6
        &HF7FB=
        &HF7FC=
```

```
        &HF7FD=

String  (&HF663=3)
        &HF7F8=  (baixo)  ender. 1 de des-
        &HF7F9=  (alto)   crição da string
        endereço 1 = comprimento da string
        endereço +1= baixo
        endereço +2= alto ...endereço 2 - a  locação  da
string
```

COMO RETORNAR UM PARÂMETRO DA LINGUAGEM DE MAQUINA

Para retornar um parâmetro de linguagem de máquina para o Basic, armazene no endereço &HF663 o valor adequado conforme o tipo de dado e armazene seus parâmetros em algum endereço após sua rotina em linguagem de máquina. O Basic encontra o lugar onde o parâmetro está aramzenado e transfere os valores em questão para as devidas variáveis, após deixar a rotina em linguagem de máquina.

CAPÍTULO 2 - O Z80A

O microprocessador Z80A é o chip de silício mais importante do seu MSX. Ele foi desenvolvido pela ZILOG INC. do Estado da Califórnia, nos Estados Unidos, e é o microprocessador de 8 bits mais sofisticado existente no mundo. Haja visto o MSX !!!

Num estudo muito aprofundado da linguagem ASSEMBLY do Z80, que não cabe neste livro, pois é quase "cultura inútil" um aprofundamento desse nível, você verá que as instruções do Z80 levam de 1 microssegundo até aproximadamente 5 microssegundos para serem executadas, considerando a frequência do "clock" do micro, em torno de 3.58 MHz. Portanto é fascinante saber que em linguagem de máquina, o micro pode executar até 3.5 milhões de instruções por segundo, correspondendo de 100 a 300 vezes mais rápida que o MSX Basic !!!

E todo o processamento do micro, apesar de possuir outros microprocessadores auxiliares, como a PPI, o PSG, ou o VDP, é efetuado pelo Z80.

Esse chip de silício possui 40 pinos, que estabelecem contato com o resto do sistema, que passaremos a denominar simplesmente de "pinos" ou "vias" . A Figura 1 mostra a disposição desses pinos na caixa do Z80. A seguir, a descrição das funções de cada um desses pinos:

- Pinos 1 a 5 e 30 a 40 (A0 até A15). Esses dezesseis pinos formam o que chamamos de vias de endereçamento (*address bus*), que são utilizados para transportar endereços do microprocessador para a memória.

- Pino 6 . Controlador do "relógio" de entrada (*CLOCK*) No MSX a freqüência desse relógio é de cerca de 3,58 MHz, ou seja, um relógio que pulsa a cada 0.000000306 de um segundo.

- Pinos 7 a 10 e 12 a 15 (D0 até D7). São oito pinos que formam as vias de dados (*Data bus*) que manipulam bytes de dados de e para o microprocessador.

- Pino 11 . Pino de voltagem (+5 Volts) - estabilizado em + 5 volts absolutos requeridos pelo microprocessador.

- Pino 16 . Pino de "Interrupções mascaradas", INT (*Interrupt Request*) - Pino que ao ser ativado, permite interrupções na execução de uma rotina em linguagem de máquina. A leitura de teclado, ou seja, a rotina que verifica se alguma tecla foi pressionada, funciona na base de interrupções.

- Pino 17 . Pino de "Interrupção não mascarada", NMI (*Non Maskable Interrupt*) - Quando este pino é ativado, ele faz com que o microprocessador pare a execução de um programa em linguagem de máquina.

- Pino 18 - pino HALT . Quando este pino está no nível 0 (Nível lógico 0 ou NL0), indica que o microprocessador está executando a instrução HALT, ou seja, entra em "estado de espera", aguardando alguma instrução. Esta instrução HALT é usada basicamente em dois casos:

1- No final de um programa, após todas as instruções em código de máquina terem sido executadas;

2- Quando é necessário permanecer com o Z80 parado, aguardando ou uma instrução ou uma interrupção.

- Pino 19 . Pino de solicitação de memória (*Memory Request*) - MREQ. Este pino é uma saída do Z80 que, quando está no nível 0, indica que existe um endereço de memória a ser utilizado nas vias de endereçamento, para leitura ou escrita da/na memória. O MREQ faz parte da seleção do chip de memória, pois informa ao meio externo que o Z80 está realizando uma leitura/escrita na memória.

- Pino 20 - Pino de Entrada/Saída - IORQ (*Input/Output Request*) . Este sinal indica que existe na metade inferior das vias de endereçamento (bits A0 até A7 - menos significativos), um endereço válido de entrada/saída (input/output), para uma operação de entrada/saída , de leitura ou escrita na memória. Um sinal IORQ também é gerado com um sinal M1, quando uma interrupção é reconhecida, para indicar que uma resposta de vetor de interrupção pode ser colocada nas vias de dados.

- Pino 21 - Pino de leitura - RD (*Memory Read*). Quando este pino está no nível 0 indica que o microprocessador quer ler dados da memória ou de algum periférico de entrada/saída.

- Pino 22 - Pino de escrita - WR (*Memory Write*) . Quando está no nível 0, indica que existe nas vias de dados D0 a D7, um byte para ser armazenado no endereço da memória ou no periférico de entrada/saída.

- Pino 23 - Pino de reconhecimento - BUSAK (*Bus Acknowledge*) . O microprocessador reconhece uma "requisição externa", interrompendo a execução de qualquer instrução e ativando este pino.

- Pino 24 - Pino de espera - WAIT . Este é um sinal de pedido de espera, com o objetivo de sincronizar memórias e periféricos de entrada/saída mais lentos que o Z80. Enquanto este sinal de entrada WAIT for mantido no nível 0, o microprocessador fica parado aguardando que o meio externo responda à sua solicitação de leitura ou escrita.

- Pino 25 - Pino de solicitação - BUSRQ (*Bus Request*). O Z80 permite que periféricos externos usem os pinos de endereçamento ou os pinos de dados, através da ativação deste pino.

- Pino 26 - Pino de inicialização - RESET . Este pino é uma entrada usada para inicializar o Z80. Ele é acionado imediatamente após ligar seu micro, ou quando se aperta o botão de RESET. Quando este pino vai para o nível 0, ocorre o seguinte:

1- Todos os pinos de endereçamentos e de dados são inicializados;

2- Todos os sinais de controle ficam inativos;

3- Os registros I e R passam para 0;

4- O modo de interrupção é colocado em 0;

5- As interrupções provenientes da entrada INT são inibidas e

6- O contador de programas é zerado (PC).

- Pino 27 - Pino de "busca" da memória - M1 (*Machine Cycle One*) . Quando vai para o nível 0, indica que está sendo executado um "fetch" (*Ciclo de busca da instrução*), da instrução corrente. Toda instrução, ao ser executada, exige que o Z80 primeiramente realize a busca do seu código de operação (*OP CODE*) que está armazenado na memória. Em seguida, o Z80 deposita este código no registro de instrução, para então

interpretá-lo.

- Pino 28 - Pino de "Restauração" da memória - RFSH (*Refresh*) . Não está diretamente relacionado com leitura ou escrita na memória. É utilizado em memórias RAM dinâmicas, como um seu refrescamento (restauração). Basicamente o RFSH é uma operação de leitura em determinadas posições da memória, sem que haja efetivamente transferência de informações. O Z80, através das vias de endereçamento (*Address Bus*), do registro R e do sinal RFSH, implementa as funções de RFSH, sem necessidade de controladores externos. Quando RFSH=0 e MREQ=0, o conteúdo do registro R é colocado nos 7 bits menos significativos (A0 a A6) das vias de endereçamento, e a cada busca de instrução (fetch), o conteúdo do registro R é incrementado em uma unidade.
- Pino 29 - Pino terra.

```
                   ------------------
A11     1  :     \          /        :  40 A10
A12     2  :      \________/         :  39 A9
A13     3  :                         :  38 A8
A14     4  :                         :  37 A7
A15     5  :                         :  36 A6
CLOCK   6  :                         :  35 A5
D4      7  :                         :  34 A4
D3      8  :                         :  33 A3
D5      9  :                         :  32 A2
D6     10  :                         :  31 A1
5 V.   11  :                         :  30 A0
D2     12  :                         :  29 GND
D7     13  :                         :  28 RFSH
D0     14  :                         :  27 M1
D1     15  :                         :  26 RESET
INT    16  :                         :  25 BUSRQ
NM1    17  :                         :  24 WAIT
HALT   18  :                         :  23 BUSAK
MREQ   19  :                         :  22 WR
IORQ   20  :_________________________:  21 RD
```

Figura 1 - Pinagem do Microprocessador Z80A.

A ESTRUTURA INTERNA DO Z8Ø

A estrutura interna desse chip é um tanto complicada, mas felizmente dividida em cinco partes que exercem funções diferentes. São elas:

1 - Unidade de controle
2 - Registro de instruções
3 - Contador de programas
4 - Registros disponíveis ao usuário
5 - Unidade lógica e aritmética

A seguir, a descrição dessas cinco partes:

UNIDADE DE CONTROLE

A unidade de controle do Z8Ø pode ser comparada a um "gerente de uma linha de produção de uma fábrica". É responsabilidade da unidade de controle e aquisição de matérias primas (bytes de dados), que são transformadas pela fábrica (estrutura do Z8Ø) em produtos finais acabados (também bytes de dados), que são enviados aos destinatários finais, garantindo assim sucesso na produção.

Essa unidade de controle produz um número muito grande de sinais de controle internos, que, através das vias de controle, vão para outras partes da estrutura interna do microprocessador, assimcomo esses sinais de controle vão para os pinos de controle RD, WR, MREQ etc.

Note porém que da mesma maneira como um gerente de uma linha de produção, esta unidade de controle não é

responsável sobre qual tarefa deva ser realizada, mas apenas como fazer.

O Z80 tem condição de funcionar como computador porque tem a habilidade de seguir um programa armazenado. Este programa deve estar presente em algum lugar da memória, de modo que ele possa ler as instruções em código de máquina, uma a uma, para então executá-las.

REGISTRO DE INSTRUÇÕES

O termo "registro" é usado para descrever um dispositivo interno do Z80 que guarda temporariamente 8 bits de um byte qualquer, para poder manipulá-los.

Nos circuitos internos deste microprocessador existem vários registros, e o movimento de bytes entre esses registros é um dos recursos mais importantes em programação em linguagem de máquina.

O registro de instruções é um registro muito especial, que armazena o código de operação de uma instrução durante todo o tempo em que esta esteja sendo executada. Esse código de operação de instrução é quem determina o que deve ser feito pelo sistema durante uma instrução.

CONTADOR DE PROGRAMA

Este contador de programa não é um registro simples de 8 bits, mas sim a junção de 2 registros de 8 bits, formando um par de registros, totalizando 16 bits, para serem usados juntos, no sentido de manter controle de endereçamentos de cada instrução guardada na memória. Sempre que uma instrução for lida da memória, a fim de ser executada, junto com ela deve ser fornecido um endereço.

O PC (*Program Counter - Contador de Programa*) terá então o endereçamento dessa instrução, sendo normalmente incrementado para, após te-la executado, apontar para a instrução seguinte.

REGISTROS DISPONÍVEIS AO USUÁRIO (REGISTROS PRINCIPAIS)

Existem, no total, 24 registros disponíveis ao usuário no microprocessador Z80. Eles são assim denominados porque podem receber e armazenar bytes de dados especificados pelo usuário. Cada registro tem seu próprio nome, que não possui lógica na maioria dos casos, e alguns deles possuem funções específicas. São registros de 1 byte, ou seja, 8 bits (daí o microprocessador ser de 8 bits - manipular 1 byte por vez), que em determinadas situações são agrupados aos pares, formando um "par de registros", capaz de manipular então, 16 bits.

A seguir a descrição dos registros disponíveis no Z80, com suas respectivas funções:

REGISTRO A (*Acumulador*)

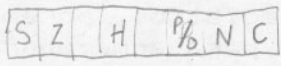
Este é o mais importante registro do Z80. Ele é mais conhecido por "acumulador", porque na maioria das operações que dele se utilizam, usam-no para acumular seus resultados.

Ele é muito usado para desenvolver operações aritméticas e lógicas, e muitas delas se utilizam apenas deste registro para atingir o resultado.

Por conseguinte, existem diversos modos pelos quais um byte de dados pode ser armazenado pelo programador neste registro. Portanto, existem muitas instruções em código de máquina que envolvem o registro A.

FLAGS: SETADAS QUANDO:

- C: CARRY → QDO HÁ VAI UM EM UMA ADIÇÃO OU EMPRÉSTA UM EM SUBTR
- P/O: PARITY/OVERFLOW → PARITY: QDO O Nº DE BITS SETADOS É PAR / OVERFLOW: QDO O Nº MUDA DE SINAL POR PASSAR DO LIMITE. -127 ou 32767
- Z: ZERO → QDO O RESULTADO DE UMA OPERAÇÃO É ZERO
- S: SINAL → QDO O RESULTADO DE UMA OPERAÇÃO É NEGATIVO

REGISTRO F

Também conhecido como "flag register", ou seja, em tradução literal, "registro de bandeiras", cujo significado é "registro de indicadores de estado". Normalmente ele é mais conhecido como uma coleção de 8 bits indicadores de algum estado específico do microprocessador, em vez de um registro propriamente dito.

O conceito desses bits indicadores de estado será visto com mais detalhes mais tarde, mas, por enquanto basta saber que determinados indicadores querem dizer algo quando estão com valor 1, e querem dizer outra coisa completamente diferente quando estão com valor Ø.

REGISTROS H e L, FORMANDO O PAR HL

Normalmente, as instruções que endereçam bytes de dados na memória, o fazem através do par de registros HL, tornando então possível o endereçamento de até 65.536 posições de memória. Lembremos que um endereço de memória sempre se utiliza de 16 bits (2 bytes) e é subdividido em "parte alta" e "parte baixa", dando origem aos nomes desses registros H (de *HIGH*) e L (de *LOW*), significando que a parte alta do endereço será armazenada no registro H e a parte baixa no registro L. Por exemplo, o endereço 7682, para ser armazenado em HL, é subdividido em 76, que vai para o registro H e 82 que vai para o registro L.

Uma memória de 65536 posições de endereçamento pode ser considerada como sendo dividida em 256 páginas de 256 posições, e, neste caso, o valor armazenado no registro H serve como indicador de qual página da memória está sendo utilizada. No microprocessador Z8Ø, o par de registros HL é um dos três pares empregados no

endereçamento de registros, porém é o mais importante. Ele pode ser usado para armazenar um número de 16 bits em vez de endereços, pois existe um grande número de operações aritméticas que podem ser realizadas com esses números.

REGISTROS B, C, D, E OU PARES BC e DE

Esses pares de registros são usados principalmente como registros de endereçamento, para auxiliar o par HL, ou utilizados individualmente, como registros comuns, embora o nome DE seja abreviação de "DESTINATION" (destino), e o registro B, quando utilizado individualmente serve como contador de loop, na maioria das vezes.

CONJUNTO DE REGISTROS ALTERNATIVOS

O conjunto de registros existentes A, B, C, D, E, F, H e L pode não ser suficiente para o programador em código de máquina. Para isso existem os registros alternativos, A', B', C', D', E', F', H', e L', que, através de instruções especiais, permitem que o conteúdo dos registros principais seja momentaneamente trocado com o conteúdo do seu equivalente alternativo. Assim, os dados anteriores ficam guardados nos registros alternativos, enquanto se trabalha com os registros principais. A cada troca, todos os registros são envolvidos, com exceção dos registros A e F.

PARES DE REGISTROS IX e IY

Esses dois pares de registros são usados em operações que envolvem indexação, que é uma facilidade que permite manipulação de itens de listagens ou tabelas, a fim de

serem pesquisadas. Esses registros mantêm um endereço base, e as posições desejadas são sempre em função desse endereço base, que deve estar obrigatoriamente armazenado no par IX ou no par IY.

REGISTRO APONTADOR DA PILHA SP (STACK POINTER)

Este registro ainda é um registro de endereçamento. Ele é usado para apontar posições na área da pilha da máquina, na memória RAM, e é sempre considerado como sendo um registro simples, porém de 2 bytes. Essa pilha da máquina (*Machine Stack*) é utilizada para se "empiulhar" endereços, da seguinte maneira: último a ser colocado, primeiro a ser retirado (LIFO = *last in, first out*), e o apontador da pilha SP é usado para armazenar o endereço da última posição a ser executada. Entretanto, quando uma nova entrada está para ser efetivada, a Unidade de Controle do microprocessador reduz o valor armazenado no ponteiro da pilha antes de aceitar essa entrada.

REGISTRO I

Este é o registro vetor de interrupção. Em sistemas baseados no Z8Ø, este registro normalmente é usado para armazenar o endereço base de uma tabela de endereços para manipulação de diferentes dispositivos de entrada/saída.

REGISTRO R

Esse é o registro de refrescamento de memória. Na realidade, ele é um simples contador que é incrementado a cada ciclo de busca de instrução. O valor no registro se altera diversas vezes entre valores de Ø a 255. O

registro R é usado para gerar parte do endereço requerido por memórias dinâmicas, de forma que possa ser "refrescado" (ou recarregado).

A UNIDADE LÓGICA E ARITMÉTICA

Este é o quinto bloco funcional do microprocessador Z80, tendo como função específica o procedimento de operações lógicas e aritméticas, de propósitos bem reduzidos; em aritmética, apenas operações de soma e subtração binárias são possíveis, e sempre de um byte contra outro byte. O usuário deverá programar rotinas em linguagem de máquinas repetitivas, caso deseje trabalhar com campos maiores que 1 byte. Por esse motivo, após cada soma ou subtração de 1 byte, um indicador do registro F, aquele chamado de CARRY FLAG, ou bandeira de transporte, poderá ser ativado (conter valor 1), simulando aquela nossa conhecida operação de "vai um", na operação aritmética dos próximos dois bits de um byte. Esta unidade também é capaz de desenvolver operações com bits, operações lógicas (AND, OR, NOT, XOR - que veremos adiante) e operações de ativar ou desativar indicadores de estados (*flags*), a fim de mostrar determinados resultados.

CAPITULO 3 — ESTRUTURA DE UM PROGRAMA EM LINGUAGEM DE MÁQUINA

Conforme já vimos, o microprocessador Z 8Ø trabalha como um computador, já que é uma pequena máquina capaz de seguir instruções de um programa armazenado. Este programa obrigatoriamente sempre será um conjunto de instruções em linguagem de máquina, associado a alguns dados, ou bytes, ou endereços da memória, tanto ROM quanto RAM, ou informações genéricas que o programa necessita, sempre armazenadas em posições consecutivas da memória.

Em microcomputadores baseados no Z8Ø, estas posições de memória armazenam no máximo 8 bits, ou 1 byte de dado. Portanto, um programa em linguagem de máquina consiste em um conjunto de dados que aparecem como uma série de números de 8 bits.

A descrição mais elementar de um programa em linguagem de máquina é a sua representação binária. Por exemplo:

ENDEREÇO	BINÁRIO
&HCØØØ	11Ø111Ø1
&HCØØ1	ØØ11ØØ11
&HCØØ2	1ØØØ1Ø11
&HCØØ3	ØØØ11ØØØ
&HCØØ4	11111ØØ1
&HCØØ5	Ø1Ø1Ø111
&HCØØ6	11ØØØØ1Ø
&HCØØ7	Ø11ØØ11Ø
&HCØØ8	1ØØØ1111

Essa representação é uma maneira perfeitamente válida para mostrar um programa em linguagem de máquina. Mas você há de convir que é dificílimo programarmos assim, e é também muito sujeito a erros. Imagine, dada essa estrutura de programa, se você trocar apenas um bit de um byte qualquer...

A seguir, o mesmo processo em linguagem de máquina, só que com outra representação, expressa em decimal e em hexadecimal. Mas, que é igualmente difícil e não muito útil, dado que não sabemos o seu significado.

ENDEREÇO	CONTEÚDO	
&HCØØØ	243	&HF3
&HCØØ1	175	&HAF
&HCØØ2	17	&H11
&HCØØ3	255	&HFF
&HCØØ4	255	&HFF
&HCØØ5	195	&HC3
&HCØØ6	2Ø3	&HCB
&HCØØ7	17	&H11
&HCØØ8	2Ø1	&HC9

Nos apêndices, no final deste livro, você encontrará, respectivamente, as conversões de valores decimais entre Ø e 255, para valores hexadecimais, e o significado dos códigos, tanto em decimal quanto em hexadecimal, para convertê-los em mnemônicas, respectivamente, as instruções do microprocessador Z8Ø. Por exemplo:

ENDEREÇO	MNEMÔNICA	COMENTÁRIOS
&HC000	DI	Desabilita interrupções
&HC001	XOR A	Efetua a operação lógica XOR com o Acumulador.
&HC002	LD DE, &HFFFF	Carrega o par DE com o valor &HFFFF
&HC005	JP &H11CB	Efetua um salto relativo para o endereco &HCB11

Nessa descrição, "mnemônica" quer dizer o nome das instruções em linguagem de máquina, que é um modo estilizado de representar a instrução, de forma que seja facilmente compreendida.

Todas as instruções em linguagem de máquina do conjunto de instruções do Z80 possuem suas próprias mnemônicas, e normalmente um programa em linguagem de máquina é descrito utilizando-se delas, em vez de valores binários, decimais ou hexadecimais.

Note na listagem acima, que duas linhas de instrução usam posições simples, enquanto outras duas linhas ocupam três posições. No último caso, a primeira posição armazena o código da instrução propriamente dito, enquanto as duas posições armazenam os dados associados a essa instrução (o operando).

A forma usual de se listar um programa em linguagem de máquina é mostrada a seguir:

ENDEREÇO	CÓDIGO HEXA	MNEMÔNICA	COMENTÁRIOS
&HC000	F3	DI	Desabilita interrupções
&HC001	AF	XOR A	Operação lógica XOR
&HC002	11FFFF	LD DE,FFFF	Endereço topo da memória, armazenado em DE
&HC005	C3CB11	JP 11CB	Equivalente ao GOTO em Basic

Essa forma de listagem normalmente é chamada de formato ASSEMBLY (montado), e possui os endereços das locações

da memória, armazenando o primeiro byte da linha de instrução, em hexadecimal; os códigos das instruções e seus dados associados (operandos), também em notação hexadecimal; as mnemônicas de cada instrução, e, finalmente, um campo para comentários, ou observações, onde o usuário pode escrever o que deseja que aconteça naquele momento.

O exemplo acima mostra como o formato assembly para um dado bloco de códigos do Z80 pode ser derivado - uma operação que normalmente é denominada DISASSEMBLY, e o programa de computador que desenvolve essas operações é conhecido por DISASSEMBLER. Portanto, programas disassembler são muito úteis, pois mostram programas em linguagem de máquina, com as posições da memória ou endereços, em hexadecimal; seus conteúdos, também em hexadecimal (alguns mostram também em decimal), e as mnemônicas das instruções correspondentes.

Em outras palavras, um programa assembler, ou montador, permite-nos desenvolver listagens em linguagem de máquina, a partir de suas mnemônicas, para que ele faça a sua conversão para seus códigos equivalentes automaticamente; e os programas chamados disassembler, ou desmontadores, permitem listarmos ("abrir uma listagem") os seus códigos e as mnemônicas das instruções correspondentes.

Às vezes, alguns programadores em linguagem de máquina, com o propósito de facilitar o seu trabalho, incluem "rótulos" (*labels*), em determinadas posições ou endereços da memória, que são constantemente atualizados durante toda a execução de uma determinada rotina.

Por exemplo:

```
INÍCIO=       &H0000
NOVO INÍCIO=  &HCB11
TOPO MEM=     &HFFFF
```

Podemos reescrever a rotina anterior, desta forma:

```
ENDEREÇO   RÓTULO   MNEMÔNICA   COMENTÁRIOS
INÍCIO
&HC000              DI          Desabilita interrupções
&HC001              XOR A       Operação XOR
&HC002              LD DE, TOPO MEM
                                LET DE= &HFFFF
&HC005              JP NOVO INÍCIO
                                GOTO &HCB11
```

A seguir, os rótulos que você frequentemente irá encontrar em programas assembler/disassembler comerciais, de acordo com a nomenclatura da ZILOG, fabricante do Z80:

RÓTULO	SIGNIFICADO
ORG XX	Coloca endereço XX no contador de referência.
EQU XX	Associa valor XX a um determinado rótulo - só pode ocorrer uma vez por rótulo.
DEFL XX	Associa o valor XX a um determinado rótulo e pode ser repetido diversas vezes com diversos valores para o mesmo rótulo.
END	Significa o final do programa-fonte. Qualquer declaração após será ignorada.
DEFT	Gera uma sequência de bytes no código objeto que representa os 7 bits do código ASCII para cada caractere da string.
EXTERNAL	Usado para declarar que cada operando seu é um símbolo definido em algum outro módulo, porém, referenciado neste.
GLOBAL	Usado para declarar que cada um dos

seus operandos é um símbolo definido neste módulo, e o nome e o valor estão disponíveis a outros módulos que contenham declarações tipo EXTERNAL para cada nome.

DEFB X — Define o conteúdo (X) de um byte no contador de referência corrente.

DEFB "X" — Define o conteúdo ("X") de um byte da memória, como sendo a representação ASCII de um caractere.

DEFW XX — Define o conteúdo de uma palavra de 2 bytes. O byte menos significativo é armazenado no contador de referência corrente, enquanto o mais significativo é armazenado no endereço anterior mais um.

A seguir alguns exemplos de listagens, que, apesar de serem completamente diferentes, querem dizer a mesma coisa.

DISASSEMBLE &HØØØØ

```
&HØØØØ  F3      DI
&HØØØ1  AF      XOR A
&HØØØ2  11FFFF  LD DE, HFFFF
&HØØØ5  C3CB11  JP 11CB
&HØØØ8  2A5D5C  LD HL, (5C5D)
&HØØØB  225F5C  LD (5C5F), HL
&HØØØE  1843    JR HØØ53
&HØØ1Ø  C3F215  JP 15F2
&HØØ13  FF      RST 38
```

Esse método é o mais comum. A primeira coluna da esquerda mostra os endereços, em valores hexadecimais, seguida dos códigos hexa, e as mnemônicas correspondentes.

Outra listagem mais simples, tabulando as informações:

```
TABULATE &H0000
&H0000  F3  AF  11  FF  FF  C3  CB  11
&H0008  2A  5D  5C  22  5F  5C  18  43
&H0010  C3  F2  15  FF  FF  FF  FF  FF
&H0018  2A  5D  5C  7E  CD  7D  00  00
```

Aqui também a primeira coluna da esquerda significa endereços, em valores hexadecimais.

Outro tipo de listagem mostra, quando possível, os caracteres ASCII correspondentes àqueles valores dos códigos hexadecimais.

```
PRINT &H0000
&H0000
&H0008  *  ]  \  "  _  \     C
&H0010
&H0018  *  ]  \  "  ’
```

E aqui, uma curiosidade: o conteúdo dos registros do Z80 e o estado das flags (bandeiras), que veremos em detalhe mais adiante.

```
READY
REGISTERS

BC      DE      HL      IX      IY      A
0000    F3B2    5E1E    03D4    5C3A    65

BC’     DE’     HL’     SP      (SP)    (HL)
0001    0000    695B    BF4D    62AC    30

BREAKPOINT : -

FLAGS = SZ-H-PNC
        01100001

READY
```

CAPÍTULO 4 — A MATEMATICA NA PROGRAMAÇAO EM LINGUAGEM DE MÁQUINA

Já vimos que num microcomputador baseado no microprocessador Z80, toda e qualquer manipulação de dados em sua memória envolve um byte de 8 bits. A melhor forma de se representar esse dado é a que utiliza a notação binária para cada bit envolvido na operação. Mas, nesse formato, os números são muito difíceis de serem manipulados, e, por essa razão, entre outras de menor importância, é que se usa a notação hexadecimal.

VALORES OU CÓDIGOS HEXADECIMAIS

O sistema hexadecimal, da mesma forma que um sistema decimal, ou outro sistema qualquer, deve ter uma base para representar valores numéricos. Conforme o próprio nome diz, a base deste sistema é dezesseis, enquanto no sistema decimal, com o qual estamos acostumados a lidar, emprega-se a base dez, e no sistema binário

emprega-se a base dois.

No nosso sistema decimal, possuímos dez valores distintos para representar quantidades diferentes. Analogamente, o sistema binário utiliza-se de dois valores, quais sejam, o zero e o um, e o sistema hexadecimal possui nada menos que dezesseis valores para sua representacao numérica. Nesta, os primeiros dez caracteres são os dígitos Ø, 1, 2, 3, 4, 5, 6, 7, 8 e 9 e os seis caracteres adicionais são as letras A, B, C, D, E, e F.

A equivalência:

DECIMAL	HEXADECIMAL	BINARIO
Ø	Ø	ØØØØ
1	1	ØØØ1
2	2	ØØ1Ø
3	3	ØØ11
4	4	Ø1ØØ
5	5	Ø1Ø1
6	6	Ø11Ø
7	7	Ø111
8	8	1ØØØ
9	9	1ØØ1
1Ø	A	1Ø1Ø
11	B	1Ø11
12	C	11ØØ
13	D	11Ø1
14	E	111Ø
15	F	1111

Note que são efetivamente dezesseis valores, apesar de terminarem em quinze!!! Começam no zero, que é o primeiro valor!!!

Mas, o que representa essa base decimal, que tanto utilizamos ? Significa que todo e qualquer valor numérico pode ser representado pelos caracteres que compõem aquela base.

Por exemplo: na base decimal, o que significa 1987 ?
Vamos decompor essa quantidade numérica:

```
 1 9 8 7
 | | | |
 | | | 7 x 1Ø ^ Ø = 7 x 1    =    7
 | | 8 x 1Ø ^ 1  = 8 x 1Ø   =   8Ø
 | 9 x 1Ø ^ 2    = 9 x 1ØØ  =  9ØØ  +
 1 x 1Ø ^ 3     = 1 x 1ØØØ  = 1ØØØ
                              ----
                              1987
```

Ou na base binária:

```
 1 1 1 Ø 1 1 1 1 Ø Ø 1 1
 | | | | | | | | | | | |
 | | | | | | | | | | | 1x2^Ø = 1 x 1 =      1
 | | | | | | | | | | 1x2^1    = 1 x 2 =      2
 | | | | | | | | | Øx2^2      = Ø x 4 =      Ø
 | | | | | | | | Øx2^3        = Ø x 8 =      Ø
 | | | | | | | 1x2^4         = 1 x 16 =    16
 | | | | | | 1x2^5           = 1 x 32 =    32
 | | | | | 1x2^6             = 1 x 64 =    64
 | | | | 1x2^7              = 1 x 128 =   128
 | | | Øx2^8                = Ø x 256 =     Ø
 | | 1x2^9                  = 1 x 512 =   512
 | 1x2^1Ø                  = 1 x 1Ø24 = 1Ø24
 1x2^11                    = 1 x 2Ø48 = 2Ø48
                                        ----
                                        3827
```

E na base hexadecimal:

```
 F A 8 B
 | | | |
 | | | B x 16 ^ Ø     =  11  x    1 =    11
 | | 8 x 16 ^ 1       =   8  x   16 =   128
 | A x 16 ^ 2         =  1Ø  x  256 =  256Ø
 F x 16 ^ 3           =  15  x 4Ø96 = 6144Ø
                                      -----
                                      64139
```

Aí estão, portanto os exemplos de conversão de bases. Eu, particularmente, hoje utilizo a calculadora gráfica CASIO 72ØØG, que converte automaticamente esses valores.

Como você deve ter notado, cada caractere hexadecimal forma uma representação binária de 4 bits. Isso significa que um byte de 8 bits é representado por um par de caracteres hexadecimais, e um número de 16 bits necessita de quatro caracteres hexadecimais.

Cada conjunto de 4 bits, denominamos NIBBLE.

Quando representamos um valor hexadecimal através da representação individual de cada caractere que o compõe, utilizamos a representação BCD (*Binary Coded Decimal*).

Por exemplo:

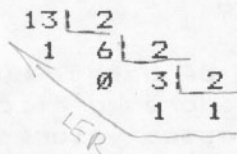
```
ØØØØØØØØb = ØØH
Ø1ØØ1111b = 4FH
ØØØØØØØØØØØØØØØØb = ØØØØH
Ø1ØØ11ØØ1Ø1Ø1111b = 4C4FH
```

A conversão de um número no sistema decimal para o

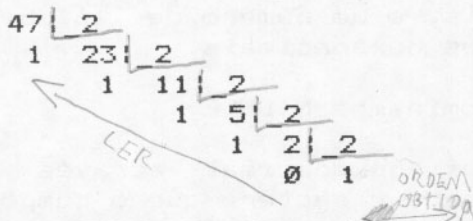
sistema binário é feita da seguinte forma: divide-se o número decimal por 2 e em seguida o quociente obtido também por 2 e assim sucessivamente, até que uma delas provoque um quociente fracionário.

O quociente da última divisão (que deve ser zero ou um), juntamente com os restos de todas as divisões, a partir da última, forma o binário equivalente ao número da base dez.

Por exemplo:

```
13|_2
 1  6|_2
    Ø  3|_2
       1  1
```

Portanto, 13d equivale a 11Ø1

Outro exemplo:

```
47|_2
 1  23|_2
     1  11|_2
         1  5|_2
            1  2|_2
               Ø  1
```

Então 47d equivale a 1Ø1111

A leitura dos números binários é feita dígito a dígito, e não como no sistema decimal. Por exemplo, 1Ø11 não é lido "um mil e onze" mas sim " um zero um um". Outro detalhe é que todo decimal par tem seu equivalente binário terminando em zero e todo decimal ímpar, com o dígito um.

Para se converter um número decimal para outro na base hexadecimal, divide-se o decimal pelo valor correspondente à potência de 16, cujo expoente

corresponde àquela posição do bit do algarismo, e converte-se o resultado para o equivalente hexadecimal, até o final. Por exemplo, para se converter 65535d em hexa:

```
1-      65535/16 = 4095.9375
2-      65535|_4096
        24575  15
         4095
        Resultado = 15d ou Fh
3-      15 x 4096 = 61440
4-      65535-61440 = 4095
5-      4095/16 = 255.9375
6-      4095|_256
        1535  15
         255
        Resultado = 15d ou Fh
7-      15 x 256 = 3840
8-      4095 - 3840 = 255
9-      255/16 = 15.9375
10-     255|_16
         95  15
         15
        Resultado = 15d ou Fh
11-     Resto final = 15d ou Fh
```

Portanto, o número 65535 corresponde a FFFF.

De posse de uma calculadora, podemos simplificar o processo. Por exemplo, vamos converter 40000d para hexa:

```
        40000|_4096
        36384  9.76...................9h
        -----
         3136|_256
         3072  12.25..................Ch
         ----
           64|_16
            0  4......................4h


                 Resto final = 0
```

Portanto, 40000d = 9C40h

ARITMETICA BINARIA

Vamos ver agora as regras para operações aritméticas binárias:

1- ADIÇÃO

```
    0 + 0 = 0
    0 + 1 = 1
    1 + 0 = 1
    1 + 1 = 0 com transporte de 1
1 + 1 + 1 = 1 com transporte de 1
```

Por exemplo:

```
8+...............1000
7................0111
--               ----
15...............1111
```

Se a soma exceder o valor decimal 15, o resultado obrigatoriamente serÁ expresso em cinco bits.

```
9+...............1001
8................1000
--               ----
17..............10001
```

2- SUBTRAÇÃO

```
0 - 0 = 0
1 - 0 = 1
1 - 1 = 0
```

```
    Ø - 1 = 1 com transporte de -1
1 - 1 - 1 = 1 com transporte de -1
```

Por exemplo:

```
14-..............111Ø
 3...............ØØ11
--               ----
11               1Ø11
```

Numa subtração, o resultado tanto pode ser positivo como negativo. A fim de se distinguir o sinal de um número, usa-se o bit mais significativo, mais a esquerda, como bit de sinal. Normalmente usa-se a convenção que atribui o valor 1 ao sinal negativo e Ø ao sinal positivo. Conseqüentemente o Z8Ø, que possui 8 bits de dados, utiliza 7 deles para armazenar o dado propriamente dito e o último para indicar o seu sinal. Muito cuidado deve ser tomado na expressão desses valores.

São necessários complexos circuitos eletrônicos para se subtrair diretamente os valores binários, e nos microcomputadores é usual realizar essas subtrações somando-se o "complemento de 2" do diminuidor ao diminuendo. Isto significa que se torna possível dispensar o circuito que executa a subtração, realizando essa operação por meio de circuitos somadores, inclusive quando se determina a forma complementar do diminuidor.

O complemento de 2 de um número binário é determinado invertendo-se cada um dos seus bits e somando-se 1 ao resultado. Por exemplo, o complemento de 2 de 76d em representação binária é determinado da seguinte maneira:

76d = Ø1ØØ11ØØ

Invertendo-se os bits:

1Ø11ØØ11

Somando-se 1:

```
10110011
      +1
---------
10110100 = -76 em forma de complemento de 2
```

3- MULTIPLICAÇÃO

Um método muito comum de realizar a multiplicação binária utiliza o princípio do deslocamento e soma dos algarismos. O multiplicando é multiplicado por cada bit do multiplicador, bit a bit, e o produto resultante obtém-se somando todos os produtos parciais convenientemente deslocados. Como os bits do multiplicador são 0 ou 1, os termos dos produtos parciais são iguais a 0 ou ao multiplicando, ou versões deslocadas deste.

Por exemplo, vamos multiplicar 193 por 21. Utilizemos para tanto um acumulador de 16 bits para guardar o resultado, e trabalhemos com representações binárias de 8 bits para o multiplicando e o multiplicador.

```
Multiplicando   193 = 11000001
Multiplicador    21 = 00010101
                      --------
                      11000001 produtos
                    11000001 parciais
                  11000001 deslocados
                  ------------
                  111111010101 soma
```

Quando se multiplicam dois números binários, o produto contém um número de bits igual à soma dos bits contidos nos dois valores envolvidos na operação. O número máximo de soma requerido para a multiplicação, usando o método de deslocamento e soma, é igual ao número de bits do multiplicador.

4- DIVISÃO

A divisão binária é realizada usando-se um método de deslocamento e subtração, ao contrário da multiplicação que acabamos de ver. Diminui-se repetidamente o divisor do dividendo, depois deste ser convenientemente deslocado, e verifica-se o sinal do resto após cada subtração. Se o sinal do resto é positivo, o valor do quociente é 1, mas se o sinal é negativo, o quociente vale Ø, e o dividendo é reconduzido ao seu valor anterior, somando-se de novo o divisor. Depois de a subtração dar um quociente positivo, ou, depois do tratamento anterior, no caso de ter dado um quociente negativo, o divisor é deslocado de uma posição, para a direita, sendo incluído o bit seguinte e repetindo-se a operação até todos os bits do divisor terem sido usados.

BIT EM (1) = VERDADE

BIT EM [Ø] = FALSIDADE

CAPÍTULO 5 — OPERAÇÕES LÓGICAS

As operações lógicas baseiam-se na ÁLGEBRA BOOLEANA, de modo que todo circuito lógico executa uma função booleana e, independentemente de sua complexidade, é formada pela utilização de portas lógicas básicas. Portanto toda a eletrônica digital, ou seja, todo o seu microcomputador baseia-se na Álgebra de Boole.

O MSX utiliza circuitos eletrônicos lógicos que produzem uma saída que depende do valor de uma ou mais variáveis de entrada. Os valores de entrada encontram-se ou no nível lógico Ø (NLØ) ou no nível lógico 1 (NL1).

Esses circuitos que realizam operações lógicas também são conhecidos como **BLOCOS** ou **PORTAS LÓGICAS** (do inglês *LOGIC GATES*). Passaremos agora a conhecer cinco operações lógicas básicas e fundamentais, a partir das quais, mediante associações, podem-se obter operações e consequentemente circuitos lógicos mais complexos. Estas operações são: **OR, NOT, AND, NAND e NOR**. Outras duas operações, derivadas da associação de duas das cinco

fundamentais, mas que nem por isso sao menos importantes são: XOR (EXCLUSIVE OR ou OR EXCLUSIVO)e a operação "COINCIDÊNCIA", qual seja, "EXCLUSIVE NOR".

OPERAÇÃO LÓGICA OR

A porta lógica OR produz uma saída lógica 1 quando pelo menos uma das entradas se encontra no nível lógico 1. Isto significa que a saída só é Ø quando todas as entradas também valem Ø.

TABELA VERDADE

ENTRADA		SAíDA
A	B	C
Ø	Ø	Ø
Ø	1	1
1	Ø	1
1	1	1

C = A + B
O sinal "+" indica a operação OR

Por exemplo:
A = Ø11ØØ1Ø1
B = 11Ø1ØØ1Ø

A + B =
Ø11ØØ1Ø1
11Ø1ØØ1Ø
____
1111Ø111

OPERAÇÃO LÓGICA NOT

A porta NOT (porta inversora ou porta complementar) produz uma saída inversa (complementar) da entrada.

TABELA VERDADE

ENTRADA	SAíDA
A	B
Ø	1
1	Ø

Por exemplo:

A = 01100101

$\bar{A}$ = 10011010 O sinal " ‾ " sobre a letra significa a operação NOT

OPERAÇÃO LÓGICA AND

A porta AND produz uma saída lógica 1 quando todas as entradas se encontram nesse mesmo nível lógico, ou seja, quando todas as entradas também valem 1; em qualquer outro caso, o resultado ou a saída é zero.

TABELA VERDADE

ENTRADA		SAÍDA
A	B	C
0	0	0
0	1	0
1	0	0
1	1	1

C = A . B O sinal "." indica a operação AND

Por exemplo:
A = 01100101
B = 11010010

A . B =

```
01100101
11010010
--------
01000000
```

OPERAÇÃO LÓGICA NAND

A porta lógica NAND é obtida combinando-se uma porta AND e uma porta NOT. Esta porta só fornece uma saída lógica 0 quando todas as entradas se encontram no nível lógico 1; em qualquer outro caso, o resultado será sempre 1.

TABELA VERDADE

ENTRADA		SAÍDA
A	B	C

0	0	1
0	1	1
1	0	1
1	1	0

$$C = \overline{A \cdot B}$$

Por exemplo:
A = 01100101
B = 11010010

$\overline{A \cdot B}$ =
01100101
11010010

10111111

OPERAÇÃO LÓGICA NOR

A porta lógica NOR é obtida combinando-se uma porta lógica OR com uma porta lógica NOT. Esta porta só produz uma saída lógica 1 quando todas as entradas possuem nível lógico 0; em todos os outros casos produz saída 0.

TABELA VERDADE

ENTRADA		SAÍDA
A	B	C
0	0	1
0	1	0
1	0	0
1	1	0

$$C = \overline{A + B}$$

Utilizando-se as entradas dos exemplos anteriores, temos que:

$\overline{A + B}$ = 00001000

OPERAÇÃO LÓGICA XOR (EXCLUSIVE OR)

Pode-se executar esta operação lógica utilizando-se um circuito de porta lógica. A porta OR exclusiva produz uma saída lógica 1 quando somente uma das entradas estiver no nível lógico 1. Em qualquer outro caso, a saída valerá 0.

TABELA VERDADE

ENTRADA		SAÍDA
A	B	C
Ø	Ø	Ø
Ø	1	1
1	Ø	1
1	1	Ø

C = A + B 1 Ø 1 O sinal "+" indica a operação XOR

OPERAÇÃO LÓGICA EXCLUSIVE NOR (PORTA COINCIDÊNCIA)

Esta porta nada mais é que a porta XOR com uma porta inversora (NOT) ligada à saída. Nestas condições, ela terá 1 como saída, quando as entradas forem iguais (quando elas coincidirem, e daí o nome); quando as entradas forem diferentes, a saída será Ø.

TABELA VERDADE

ENTRADA		SAÍDA
A	B	C
Ø	Ø	1
Ø	1	Ø
1	Ø	Ø
1	1	1

C = A . B O sinal " . " indica a operação EXC. NOR

CAPÍTULO 6 — O CONJUNTO DE INSTRUÇÕES DO Z80A

Essas instruções, que a partir deste capítulo passaremos a discutir e, posteriormente, aplicar em exemplos práticos, são divididas em 18 grupos, onde cada instrução tem alguma semelhança com as outras. Porém, antes de discuti-las, devemos mencionar seis classes de dados que podem completar uma instrução do Z80.

Essas classes são:

1- Um número, na faixa de um byte simples, ou seja, na faixa de 0 até 255, ou &H00 até &HFF. As instruções que requeiram um byte terão sua indicação seguida de "dd". Por exemplo, a mnemônica "LD D, dd".

2- Um número de dois bytes, na faixa de 0 a 65535, ou &H00 até &HFFFF, representado por "dddd", como na mnemônica "LD BC, dddd".

3- Um endereço (2 bytes), na mesma faixa numérica que a anterior, porém com a representação junto com as

mnemônicas de "end", como por exemplo, a instrução JP "end".

4- Um deslocamento de um byte, ou seja, um número na faixa idêntica a de um byte, considerado obrigatoriamente em sua forma de complemento de dois aritmético, com representação após a mnemônica de "e", como, por exemplo, a instrução "JR e".

5 - Um byte para deslocamento indexado, também na faixa numérica de um byte, e aqui nesta classe, também considerado em sua forma de complemento de dois aritmético, com representação nas instruções que requeiram este tipo de deslocamento de "d", como, por exemplo, "LD A, (IX+d)".

6- Um byte para deslocamento indexado e um byte simples na faixa numérica de -128d ate +127d, para o primeiro byte, e na faixa numérica de Ød a 255d, para o segundo valor, com as respectivas representações de "d" e "dd", como na instrução LD (IX+d),dd.

Passemos então aos capítulos sobre as instruções.

Coragem - o caminho é longo mas compensa.

CAPÍTULO 7 - INSTRUÇÕES DE NÃO OPERAÇÃO

MNEMÔNICA CÓDIGO HEXA
NOP 00

Esta instrução NOP, quando executada pelo microprocessador, faz com que este interrompa suas atividades por cerca de 1.14 microssegundos. Nenhum dos registros ou indicadores são afetados por esta instrução.

Esta instrução é muito útil em casos de se querer uma pausa determinada em certas rotinas de linguagem de máquina, ou para cancelar ou apagar instruções em linguagem de máquina em rotinas já prontas (como aquela de seu programa predileto, que exibe a mensagem de COPYRIGHT...).

CAPÍTULO 8 — INSTRUÇÕES DE REGISTROS COM VALORES NUMÉRICOS

As instruções a seguir são desenvolvidas no sentido de carregar registros com simples bytes, constantes.

MNEMÔNICA	CÓDIGO HEXA
LD A, dd	3Edd
LD H, dd	26dd
LD L, dd	2Edd
LD B, dd	Ø6dd
LD C, dd	ØEdd
LD D, dd	16dd
LD E, dd	1Edd

Como se pode notar pelos códigos hexa, estas instruções requerem duas locações da memória, uma para o código da instrução e outra para o valor envolvido, o operando.

As instruções acima podem ser consideradas análogas às instruções de atribuição em Basic, onde se atribui um valor qualquer a uma variável de nome conhecido. No nosso caso, atribuímos a um registro determinado o valor

de um byte especificado, ou, em outros termos, carregamos aquele registro com o conteúdo especificado. Este valor será então armazenado no registro, anulando o anterior.

As instruções a seguir são análogas às anteriores, porém envolvem pares de registros e valores de dois bytes.

MNEMÔNICA	**CÓDIGO HEXA**
LD HL, dddd	21dddd
LD BC, dddd	Ø1dddd
LD DE, dddd	11dddd
LD IX, dddd	DD21dddd
LD IY, dddd	FD21dddd
LD SP, dddd	31dddd

Estas instruções vão requerer três ou quatro locações na memória.

O primeiro byte do valor numérico da instrução vai sempre para o registro menos significativo do par de registros envolvido na instrução, enquanto o segundo byte, logicamente, vai para o outro registro envolvido, o registro mais significativo do par. Entende-se por registro menos significativo os registros **L, C, E, X, Y** e **P**,e por registro mais significativo, os registros **H, B, D, I** e **S.**

Também essas instruções são análogas às instruções de atribuição de valores da linguagem Basic. Aqui, elas armazenam naquele par de registros envolvido, um valor especificado.

As instruções deste grupo não afetam as bandeiras indicadoras de estado.

CAPÍTULO 9 - INTRUÇÕES DE COPIAR E TROCAR CONTEÚDOS DE REGISTROS

São ao todo 59 instruções do universo de instruções do microprocessador Z80, que tratam de copiar conteúdos de registros ou par de registros. Estas instruções podem ser subdivididas em 4 subgrupos.

SUBGRUPO 1 - INSTRUÇÕES DE COPIAR CONTEÚDOS DE REGISTROS SIMPLES

A tabela a seguir fornece o código das instruções que tratam da cópia de conteúdos de registros simples, genericamente denominados registros "r", para outros registros especificados.

REG. R	LD A,r	LD H,r	LD L,r	LD B,r	LD C,r	LD D,r	LD E,r
A	7F	67	6F	47	4F	57	5F
H	7C	64	6C	44	4C	54	5C
L	7D	65	6D	45	4D	55	5D
B	78	6Ø	68	4Ø	48	5Ø	58
C	79	61	69	41	49	51	59
D	7A	62	6A	42	4A	52	5A
E	7B	63	6B	43	4B	53	5B

Nenhuma das instruções contidas nessa tabela afetam as flags. Existem ainda quatro instruções envolvendo os registros I e R:

MNEMÔNICA	CÓDIGO HEXA
LD A, I	ED57
LD A, R	ED5F
LD I, A	ED47
LD R, A	ED4F

Essas instruções afetam a flag de paridade ou excesso (mais adiante você vai saber o que faz esta flag).

SUBGRUPO 2 - INSTRUÇÕES DE COPIAR CONTEÚDOS DE PAR DE DE REGISTROS

São apenas três instruções neste sub-grupo, e todas envolvem o par de registros de função especial, chamado "ponteiro ou apontador da pilha" (*stack pointer*).

MNEMÔNICA	CÓDIGO HEXA
LD SP, HL	F9
LD SP, IX	DDF9
LD SP, IY	FDF9

Estas instruções não afetam as flags.

Note que não existem instruções para copiar conteúdos de pares de registros genéricos e, portanto, as instruções acima não são apropriadas para o caso. Esta operação é desenvolvida com duas instruções de carregamento e cópia de registros simples.

Por exemplo, para se desenvolver a operação LD HL, DE, utilizamos primeiramente LD H, D e em seguida LD L, E.

Como alternativa, que consome mais memória e mais tempo de execução, o conteúdo do primeiro par de registros pode ser armazenado na pilha da máquina, para em seguida ser copiado no segundo par (veja instruções da pilha).

SUBGRUPO 3 - INSTRUÇÃO EX DE, HL

MNEMÔNICA	CÓDIGO HEXA
EX DE, HL	EB

Esta instrução, muito útil por sinal, permite que o programador troque o conteúdo do par de registros DE com o conteúdo do par de registros HL, sem afetar qualquer

flag, com grande velocidade e economia de memória, já que ocupa apenas um byte.

Esta instrução é normalmente utilizada quando um endereço ou um valor numérico que ocupa dois bytes deve ser movido do par de registros DE para o par HL, mas sem cancelar ou perder o conteúdo original de HL.

SUBGRUPO 4 - INSTRUÇÕES COM O GRUPO ALTERNATIVO DE REGISTROS

MNEMÔNICA	CÓDIGO HEXA
EXX	D9
EX AF, AF'	Ø8

A instrução EXX faz com que o conteúdo dos registros H, L, B, C, D e E sejam trocados respectivamente com o conteúdo dos registros H', L', B', C', D' e E'. A instrução EX AF, AF', conforme o seu nome sugere, faz a troca de conteúdos entre AF e AF'.

Estes registros alternativos são sempre utilizados para armazenar endereços ou valores numéricos, protegendo estes valores contra qualquer erro ou acidente no decorrer do programa, podendo ser utilizados a qualquer momento, de um modo muito rápido e fácil.

CAPÍTULO 1Ø — INSTRUÇÕES PARA CARREGAMENTO DE REGISTROS COM VALORES NUMÉRICOS COPIADOS DE ENDEREÇOS DA MEMÓRIA

O conjunto das instruções do Z8Ø possui muitas instruções que procuram dados em endereços da memória, para então carregá-los em determinados registros.

Todas estas instruções requerem do programador a especificação do endereço, ou endereços, de onde os dados devam ser copiados, para então, o registro, ou o par de registros, receber estes dados.

As instruções desse grupo podem ser subdivididas em três subgrupos, dependendo da técnica de endereçamento selecionada pelo programador.

Estas técnicas de endereçamento são:

1- **Endereçamento absoluto** - o endereço atual de dois bytes é especificado segundo sua própria instrução.

2- **Endereçamento indireto**- o endereço de dois bytes

está sempre disponível em algum par de registros de endereçamento.

3- Endereçamento indexado- o endereço da locação "a" deve ser computado pela adição de um valor de deslocamento, "d", ao endereço base armazenado ou no par IX ou no par IY.

SUBGRUPO 1 - INSTRUÇÕES UTILIZANDO ENDEREÇAMENTO ABSOLUTO

MNEMÔNICA	CÓDIGO HEXA	
LD A, (END)	3A END	
LD HL, (END)	2A END	forma usual
	ED6B END	forma não usual
LD BC, (END)	ED4B END	
LD DE, (END)	ED5B END	
LD IX, (END)	DD2A END	
LD IY, (END)	FD2A END	
LD SP, (END)	ED7B END	

A instrução LD A, (END) é a única instrução do conjunto de instruções do Z8Ø que permite carregar, em endereçamento direto ou absoluto, o conteúdo especificado daquela locação da memória em um registro simples.

Note que as instruções remanescentes deste grupo podem ser consideradas como sendo instruções duplas, ou seja, por exemplo, a instrução LD BC, (END) pode ser considerada como primeiramente LD C, (END) seguida de LD B, (END+1).

Repare que o uso do parênteses significa o endereço apontado por aquela posição e não aquela posição. Por exemplo, LD BC, XXXX significa carregar o par BC com o valor XXXX e LD BC, (XXXX) significa carregar o par BC com os valores armazenados nas posições apontadas por XXXX.

Em qualquer caso, o conteúdo do endereço especificado é copiado no registro menos significativo, e o conteúdo do endereço seguinte e copiado no registro mais significativo.

SUBGRUPO 2 - INSTRUÇÕES QUE UTILIZAM ENDEREÇAMENTO INDIRETO

MNEMÔNICA	CÓDIGO HEXA
LD A, (HL)	7E
LD A, (BC)	ØA
LD A, (DE)	1A
LD H, (HL)	66
LD L, (HL)	6E
LD B, (HL)	46
LD C, (HL)	4E
LD D, (HL)	56
LD E, (HL)	5E

Em todos os casos, o endereço da locação de onde o byte será copiado deve estar presente obrigatoriamente em algum par de registro entre HL, BC ou DE.

Note que, por exemplo, a instrução LD D, (BC) não existe, e, portanto, devem ser executadas outras instruções que efetuem o mesmo processamento:

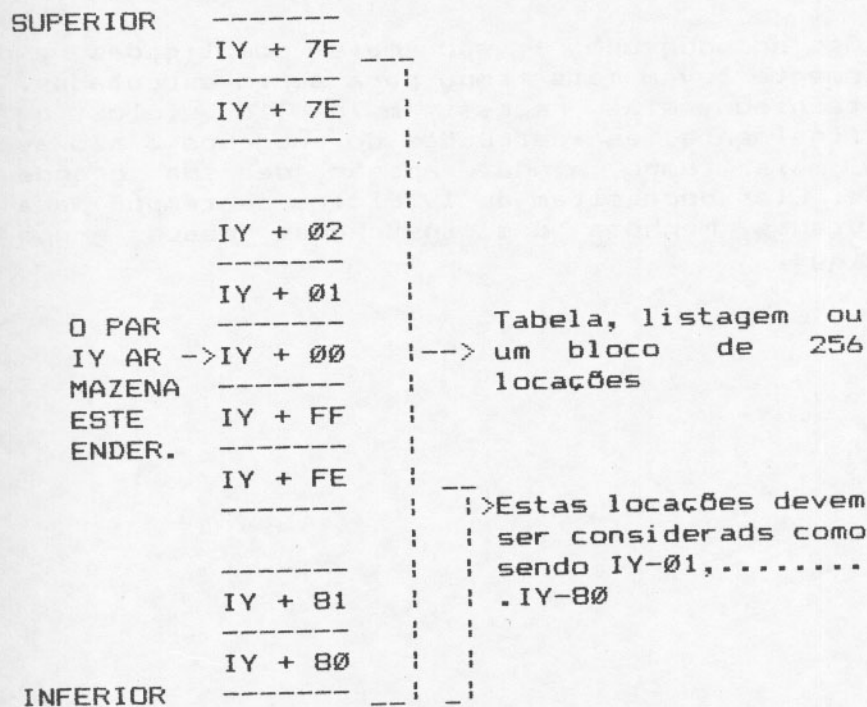
1- LD A, (BC) seguida de LD D, A, que alterará o conteúdo do registro A, ou

2- LD H, B seguida de LD L, C e finalmente LD D, (HL) que alterará o conteúdo de HL.

SUBGRUPO 3 - INSTRUÇÕES USANDO ENDEREÇAMENTO INDEXADO

As instruções deste subgrupo permitem ao programador carregar simples bytes de dados, em registros simples, armazenados sob a forma de listagens, tabelas, ou apenas blocos de dados. O endereço base é armazenado no par de registros de indexação apropriado.

NOTA: Você já deve ter notado que as instruções que envolvem os pares de registros IX e IY diferem apenas no código inicial, ou seja, para o par IX utiliza-se o código DD, e para o par IY utiliza-se o código FD. Portanto, nas instruções a seguir, envolvendo o par IX, substitua os códigos DD, por FD, ao utilizar o par IY.

O diagrama abaixo ilustra esse efeito:

MNEMÔNICA	CÓDIGO HEXA
LD A, (IX +D)	DD7E D
LD H, (IX +D)	DD66 D
LD L, (IX +D)	DD6E D
LD B, (IX +D)	DD46 D
LD C, (IX +D)	DD4E D
LD D, (IX +D)	DD56 D
LD E, (IX +D)	DD5E D

Não se esqueça das instruções que envolvem o par IY.

É interessante notar o tempo que o microprocessador Z80 leva para executar essas instruções. As instruções mais rápidas são as que compõem o subgrupo 2, que requerem do Z80 a busca de um simples byte de código e, então, o byte seguinte de dado.

As instruções do subgrupo 1 são mais complicadas e conseqüentemente levam mais tempo para serem executadas. Em termos técnicos, elas necessitam de 16 ciclos de tempo, e, finalmente, as instruções do subgrupo 3 são as que levam mais tempo ainda, apesar de sua grande praticidade. Elas necessitam de 19 ciclos de tempo para serem executadas. Nenhuma das instruções deste grupo afeta as flags.

CAPÍTULO 11 — INSTRUÇÕES PARA ARMAZENAR DADOS COPIADOS DE REGISTROS, OU VALORES NUMÉRICOS EM ENDEREÇOS OU LOCAÇÕES DA MEMÓRIA

De uma maneira geral, as instruções deste grupo efetuam operações opostas às do grupo anterior.

Estas instruções permitem que conteúdos de registros especificados pelo programador sejam copiados em endereços específicos da memória, ou que valores numéricos sejam armazenados nesses endereços. Outra vez, estas instruções são mais bem analisadas, se divididas em três subgrupos:

SUBGRUPO 1 — INSTRUÇÕES UTILIZANDO ENDEREÇAMENTO ABSOLUTO

MNEMÔNICA	CÓDIGO HEXA
LD (END), A	32 END

LD (END), HL	22 END ou ED63 (não usual)
LD (END), BC	ED43 END
LD (END), DE	ED53 END
LD (END), IX	DD22 END
LD (END), IY	FD22 END
LD (END), SP	ED73 END

As instruções acima são as únicas a utilizar endereçamento absoluto, e é importante notar que não existe uma instrução para carregar um endereço específico com um valor numérico. Se o programador necessita efetuar essa operação, deve fazê-lo, carregando primeiramente o registro A, com o valor especificado, para então armazenar o valor desejado no endereço especificado.

Novamente uma instrução do tipo LD (END), HL é na realidade uma instrução dupla, pois requer LD (END), L e então LD (END+1), H. As instruções deste subgrupo são muito utilizadas para armazenar endereços e números em locações da memória, quando estes valores são considerados variáveis.

SUBGRUPO 2 - INSTRUÇÕES QUE USAM ENDEREÇAMENTO INDIRETO

As instruções deste subgrupo permitem ao programador a cópia de conteúdos de registros simples em endereços da memória que estejam armazenados nos pares de registros HL, BC ou DE. Também existe uma instrução para carregar um byte em uma locação endereçada pelo par de registros HL.

MNEMÔNICA	CÓDIGO HEXA
LD (HL), A	77
LD (BC), A	02
LD (DE), A	12
LD (HL), H	74
LD (HL), L	75
LD (HL), B	70

```
LD (HL), C       71
LD (HL), D       72
LD (HL), E       73
LD (HL), dd      36 dd
```

SUBGRUPO 3 - INSTRUÇÕES UTILIZANDO ENDEREÇAMENTO INDEXADO

MNEMÔNICA	CÓDIGO HEXA
LD (IX+D), A	DD77 D
LD (IX+D), H	DD74 D
LD (IX+D), L	DD75 D
LD (IX+D), B	DD7Ø D
LD (IX+D), C	DD71 D
LD (IX+D), D	DD72 D
LD (IX+D), E	DD73 D
LD (IX+D), dd	DD36 dd

Novamente, para instruções que envolvam o par de registros IY, mude o código DD para FD e IX para IY.

CAPÍTULO 12 - INSTRUÇÕES DE ADIÇÃO

Este grupo é o primeiro dos quatro grupos do conjunto de instruções do Z80 que envolvem operações aritméticas ou lógicas.

As instruções de adição permitem ao programador adicionar, em aritmética binária absoluta, um número especificado ao conteúdo de um registro simples, ou ao conteúdo de um par de registros, ou ainda, a um endereço indexado da memória.

As instruções deste grupo podem ser subdivididas em três subgrupos, de acordo com suas mnemônicas.

Estes três subgrupos são:

1- Instruções de adição - **ADD**

2- Instruções de incremento - **INC**, ou seja, um caso especial da adição, quando apenas uma unidade é adicionada a um número existente.

3- Instruções de adição, porém considerando-se o estado da flag de transporte (CARRY FLAG) - **ADC**.Esta flag de transporte é um bit do registro F, utilizado para nos avisar se houve aquele nosso conhecido "vai um" nas operações aritméticas de soma.

As instruções de adição, bem como as instruções de adição com transporte, afetam a flag de transporte, mas as instruções de incremento não, fato esse que em algumas situações oferece vantagem.

SUBGRUPO 1 - INSTRUÇÕES ADD

MNEMÔNICA	CÓDIGO HEXA
ADD A, dd	C6 dd
ADD A, A	87
ADD A, H	84
ADD A, L	85
ADD A, B	80
ADD A, C	81
ADD A, D	82
ADD A, E	83
ADD A, (HL)	86
ADD A, (IX+D)	DD86 d
ADD A, (IY+D)	FD86 d
ADD HL, SP	39
ADD IX, IX	DD29
ADD IX, BC	DD09
ADD IX, DE	DD19
ADD IX, SP	DD39

Para instruções que envolvem o par de registros IY, substitua IX por IY e DD por FD.

SUBGRUPO 2 - INSTRUÇÕES INC

As instruções deste grupo permitem que o conteúdo de um simples registro, ou de um par de registros, ou mesmo de uma locação da memória seja incrementado em uma unidade. Em todos os casos, a flag de transporte é ignorada.

MNEMÔNICA	CÓDIGO HEXA
INC A	3C
INC H	24
INC L	2C
INC B	Ø4
INC C	ØC
INC D	14
INC E	1C
INC (HL)	34
INC (IX+D)	DD34 d
INC (IY+D)	FD34 d
INC HL	23
INC BC	Ø3
INC DE	13
INC SP	33
INC IX	DD23
INC IY	FD23

SUBGRUPO 3 - INSTRUÇÕES ADC

MNEMÔNICA	CÓDIGO HEXA
ADC A, dd	CE dd
ADC A, A	8F
ADC A, H	C
ADC A, L	8D
ADC A, B	88
ADC A, C	89
ADC A, D	8A
ADC A, E	8B

```
ADC A, (HL)      8E
ADC A, (IX+D)    DD8E d
ADC A, (IY+D)    FD8E d
ADC HL, HL       ED6A
ADC HL, BC       ED4A
ADC HL, DE       ED5A
ADC HL, SP       ED7A
```

As instruções deste sub-grupo permitem ao programador adicionar dois números, juntamente com o estado da flag de transporte, pois todas as instruções do grupo, conforme suas próprias mnemonicas dizem, afetam essa bandeira. Esta será RESETADA (Ø) se a instrução ADC em execução não der excesso, e será SETADA (1), se houver excesso (vai um) na operação corrente.

Faça alguns exercícios, a título apenas de prática, somando valores com o "vai um", e compare os resultados com as operações lógicas binárias.

NOTA: Nada impede que a flag de transporte seja considerada um "nono" bit do acumulador.

CAPÍTULO 13 — INSTRUÇÕES DE SUBTRAÇÃO

As instruções de subtração permitem que o programador subtraia, em aritmética binária absoluta, um número especificado do conteúdo de um registro simples, de um par de registros ou de uma locação endereçada da memória.

Novamente este grupo pode ser subdividido em três subgrupos, conforme suas mnemônicas:

1- Instruções **SUB** - Subtração simples

2- Instruções **DEC** - Casos especiais de subtração onde um número específico é decrementado de uma unidade.

3- Instruções **SBC** - O valor da flag de transporte também é subtraído do resultado.

Deste grupo, apenas as instruções DEC não afetam a flag de transporte.

SUBGRUPO 1 - INSTRUÇÕES SUB

MNEMÔNICA	CÓDIGO HEXA
SUB dd	D6 dd
SUB A	97
SUB H	94
SUB L	95
SUB B	9Ø
SUB C	91
SUB D	92
SUB E	93
SUB (HL)	96
SUB (IX+D)	DD96 D
SUB (IY+D)	FD96 D

NOTA: As mnemônicas para as instruções de subtração SUB são normalmente escritas na forma acima, ou seja, "SUB L", seria o mesmo que escrever "SUB A, L", pois todas elas envolvem o acumulador. Portanto, o "A" do nome, referindo-se ao registro A é omitido.

No Z8Ø, as instruções de subtração fornecem uma subtração binária absoluta "verdadeira".

A flag de transporte é resetada se o valor original do registro A é "maior que" ou "igual" ao subtraendo (o segundo número da subtração), e é setada se o valor do registro A é "menor que" o subtraendo.

SUBGRUPO 2 - INSTRUÇÕES DEC

As instruções deste subgrupo permitem que seja subtraída uma unidade do conteúdo de um registro simples de 8 bits, ou de um par de registros, ou de um endereço da memória.

Em qualquer caso, a flag de transporte não é afetada.

MNEMÔNICA	CÓDIGO HEXA
DEC A	3D
DEC H	25
DEC L	2D
DEC B	Ø5
DEC C	ØD
DEC D	15
DEC E	1D
DEC (HL)	35
DEC (IX+D)	DD35 D
DEC (IY+D)	FD35 D
DEC HL	2B
DEC BC	ØB
DEC DE	1B
DEC SP	3B
DEC IX	DD2B
DEC IY	FD2B

SUBGRUPO 3 – INSTRUÇÕES SBC

MNEMÔNICA	CÓDIGO HEXA
SBC A, dd	DE dd
SBC A, A	9F
SBC A, H	9C
SBC A, L	9D
SBC A, B	98
SBC A, C	99
SBC A, D	9A
SBC A, E	9B
SBC A, (HL)	9E
SBC A, (IX+D)	DD9E D
SBC A, (IY+D)	FD9E D
SBC HL, HL	ED62
SBC HL, BC	ED42
SBC HL, DE	ED52
SBC HL, SP	ED72

Uma operação SBC efetuará uma subtração binária verdadeira se a flag de transporte estiver resetada, mas executará uma subtração, considerando o "empresta um" se a flag de transporte estiver setada.

CAPÍTULO 14 — INSTRUÇÕES DE COMPARAÇÃO

As instruções deste grupo são usadas muito freqüentemente em qualquer rotina em linguagem de máquina. Elas permitem que o programador compare o valor armazenado no registro A, com uma constante, um valor armazenado em um registro qualquer de um endereço da memória.

Uma instrução de comparação efetua uma operação de subtração, sem transporte, mas descarta a resposta após usá-la, para setar as devidas bandeiras indicadoras do registro F. O valor original do registro A permanece inalterado.

A flag de transporte é afetada da mesma maneira que numa operação de subtração. Uma comparação que seja "maior que" ou "igual a" RESETA a flag de transporte, e uma comparação que seja "menor que" SETA a flag de transporte.
*
As instruções deste grupo são instruções de comparação

* SE FOR IGUAL, SETA A FLAG ZERO, SENÃO RESETA.

simples, e as instruções de comparações de blocos serão consideradas mais tarde.

MNEMONICA	CODIGO HEXA
CP dd	FE dd
CP A	BF
CP H	BC
CP L	BD
CP B	B8
CP C	B9
CP D	BA
CP E	BB
CP (HL)	BE
CP (IX+D)	DDBE D
CP (IY+D)	FDBE D

Essas instruções desenvolvem processamento análogo às instruções em Basic, de operações e decisões lógicas, tipo IF...THEN...

A >= XX -> ZERA CARRY

A < XX -> SETA CARRY

A = XX -> SETA FLAG ZERO

CAPÍTULO 15 - INSTRUÇÕES LÓGICAS

No conjunto de instruções do Z80 existem instruções para processarem operações AND, OR e XOR, que comparam o conteúdo do acumulador com o conteúdo de outra locação especificada. A operação é desenvolvida bit a bit, e o resultado de 8 bits é armazenado no acumulador.

Dividimos este grupo em três subgrupos, de acordo com suas mnemônicas.

SUBGRUPO 1 - INSTRUÇÕES AND

A operação lógica AND efetuada entre dois bits dará como resultado um bit com valor 1 somente se os dois bits envolvidos na operação valerem 1 também. Em qualquer outro caso o resultado será Ø.

MNEMÔNICA	CÓDIGO HEXA
AND dd	E6 dd

AND A	A7 → LIMPA CARRY FLAG P/ SBC HL,DE
AND H	A4
AND L	A5
AND B	AØ
AND C	A1
AND D	A2
AND E	A3
AND (HL)	A6
AND (IX+D)	DDA6 D
AND (IY+D)	FDA6 D

Ao usar uma instrução AND, todos os bits do acumulador serão RESETADOS. Esse processo permite ao programador controlar certos bits de um byte de dados.

SUBGRUPO 2 - INSTRUÇÕES OR

A operação lógica OR, executada entre dois bits, dará como resultado um terceiro bit, valendo 1, se apenas um ou até os dois bits envolvidos valerem 1.

MNEMÔNICA	CÓDIGO HEXA
OR dd	F6 dd
OR A	B7
OR H	B4
OR L	B5
OR B	BØ
OR C	B1
OR D	B2
OR E	B3
OR (HL)	B6
OR (IX+D)	DDB6 D
OR (IY+D)	FDB6 D

SUBGRUPO 3 - INSTRUÇÕES XOR

A operação lógica XOR, desenvolvida entre dois bits,

dará como resultado um terceiro bit, que valerá 1 se apenas um e somente um dos bits envolvidos valer 1 também; caso contrário , o terceiro bit valerá Ø.

MNEMÔNICA	CÓDIGO HEXA
XOR dd	EE dd
XOR A	AF
XOR H	AC
XOR L	AD
XOR B	A8
XOR C	A9
XOR D	AA
XOR E	AB
XOR (HL)	AE
XOR (IX+D)	DDAE D
XOR (IY+D)	FDAE D

Ao se utilizar a instrução XOR, os bits do registro A serão alterados se necessário for.

O uso dessas instruções, em rotinas em linguagem de máquina, é um pouco complicado, mas a instrução XOR A é freqüentemente usada como uma alternativa para LD A, Ø. Ambas as instruções limpam o acumulador, mas a primeira utiliza apenas um endereço, enquanto a segunda consome duas locações da memória.

CAPÍTULO 16 — INSTRUÇÕES DE SALTO (*JUMP*) E ESTUDO DAS FLAGS (BANDEIRAS INDICADORAS DE ESTADO DO REGISTRO F)

Neste grupo, temos no total 17 instruções de salto, que permitem que o programador efetue saltos dentro de um programa. Um salto em linguagem de máquina pode ser comparado à instrução Basic "GOTO". As instruções deste grupo são mais bem analisadas se divididas em sete subgrupos.

Quatro desses subgrupos contêm instruções condicionais que dependem do estado dos bits do registro F, que são as flags.

SUBGRUPO 1 - INSTRUÇÕES DE SALTO ABSOLUTO (JP DE JUMP)

Esta é a instrução clássica de salto. Quando a instrução JP END é executada, permite que o endereço especificado seja armazenado no PC (*Program Counter* - contador de

programa), fazendo com que a execução do programa continue a partir daquele endereço.

MNEMÔNICA	CÓDIGO HEXA
JP END	C3 END

SUBGRUPO 2 - INSTRUÇÕES DE SALTO QUE UTILIZAM ENDEREÇAMENTO INDIRETO

Estas três instruções permitem que um byte devidamente armazenado no par de registros especificado seja carregado no PC. As instruções deste subgrupo são freqüentemente utilizadas quando deve ser feito um salto para uma locação especificada em uma tabela de endereços.

MNEMÔNICA	CÓDIGO HEXA
JP (HL)	E9
JP (IX)	DDE9
JP (IY)	FDE9

SUBGRUPO 3 - INSTRUÇÕES DE SALTO RELATIVO (JR DE JUMP RELATIVE)

MNEMÔNICA	CÓDIGO HEXA
JR e	18 e

Esta instrução permite que o programador salte 127 endereços para a frente (positivos) e 128 endereços para trás (negativos), a partir do endereço corrente.

Note que o endereço corrente é de fato o seguinte ao deslocamento "e", especificado na instrução.

O deslocamento "e" é sempre considerado, em aritmética complemento de dois, e um deslocamento "e" positivo dá o número de locações que devem ser saltadas acima,

enquanto um deslocamento "e" negativo mostra em quanto o contador de programas deve ser reduzido.

SUBGRUPO 4 - INSTRUÇÕES DE SALTO CONDICIONAL RELATIVAS AO ESTADO DA BANDEIRA INDICADORA DE TRANSPORTE (*CARRY FLAG*)

São quatro instruções neste subgrupo, que permitem que seja feito um salto somente se a flag de transporte estiver no estado especificado pela instrução.

Agora vamos ver o que realmente é esta bandeira indicadora de transporte.

Esta bandeira é o bit Ø do registro F e é essencialmente um indicador que mostra se houve ou não um excesso numa operação binária, ou seja, se ela está setada em certas situações e resetada em outras ocasiões. Em muitas situações, a flag de transporte não é afetada pela execução de algumas instruções.

Em resumo:

1- Todas as instruções ADD e ADC afetam esta flag. Se não houver excesso a flag será resetada, mas se houver a flag será setada.

2- Todas as instruções SUB, SBC e CP afetam também esta flag.

3- Instruções como AND, OR e XOR resetam a flag.

4- As instruções de rotação, que veremos adiante, também afetam esta flag.

As instruções deste subgrupo são:

MNEMONICA	CODIGO HEXA
JP NC, END	D2 END
JR NC, e	3Ø e

JP C, END DA END
JR C, e 38 e

Nas duas primeiras instruções, o salto somente será executado se a flag de transporte estiver resetada.

Nas duas últimas, o salto será executado se a flag de transporte estiver setada.

SUBGRUPO 5 - INSTRUÇÕES DE SALTO CONDICIONAL RELATIVAS AO ESTADO DA FLAG ZERO

Esta bandeira zero é o bit 6 do registro F e muitas vezes indica se o resultado de uma determinada operação foi zero (quando ela estiver setada, ou seja, valer 1) ou se o resultado de uma operação foi diferente de zero (quando então ela estiver resetada, quer dizer, valer Ø).

Resumindo:

1- Instruções ADD, INC, ADC, SUB, DEC, SBC, CP, AND, OR e XOR usando registros simples de um byte, e as instruções ADC e SBC usando pares de registros irão setar a bandeira zero, se o resultado da operação em questão for zero.

2- Instruções de rotação, ou instruções de testar determinados bits (que também veremos mais adiante), ou instruções de procura de blocos afetam a flag zero.

3- Instruções LD, com exceção de LD A, I e LD A, R não afetam esta flag.

Neste subgrupo são quatro instruções que permitem que seja efetuado um salto apenas se o estado da flag zero coincidir com o especificado pela instrução.

MNEMÔNICA CÓDIGO HEXA

```
JP NZ, END      C2 END
JR NZ, e        20 e
JP Z, END       CA END
JR Z, e         28 e
```

Como o subgrupo anterior, as duas primeiras instruções permitem que seja considerado o salto apenas se a flag zero estiver resetada, enquanto nas duas últimas, o salto somente será executado se a flag zero estiver setada.

SUBGRUPO 6 - INSTRUÇÕES DE SALTO CONDICIONAL RELATIVAS AO ESTADO DA FLAG DE SINAL (*SIGNAL FLAG*)

Esta flag é o bit 7 do registro F, e em muitos casos é uma cópia do bit mais à esquerda do resultado.

Sempre que um número de 8 bits, ou um número de 16 bits é considerado na sua forma de complemento de 2 aritmético, então o bit mais à esquerda, ou seja, o bit 7 ou o bit 15 é considerado como bit de sinal.

Atenção - Este bit de sinal será resetado para números positivos e setado para números negativos.

Resumindo:

1- Instruções ADD, INC, ADC, SUB, DEC, SBC, CP, AND, OR e XOR usando registros simples de 8 bits, e as instruções ADC e SBC usando pares de registros afetam a flag de sinal.

2- Instruções de busca de blocos e muitas instruções de rotação também afetam esta flag.

3- Instruções LD, com exceção de LD A, I e LD A, R não afetam esta flag.

Neste subgrupo são duas instruções que permitem que seja

realizado um salto somente se o estado da flag de sinal coincidir com a especificação da instrução.

MNEMÔNICA	CÓDIGO HEXA
JP P, END	F2 END
JP M, END	FA END

Na primeira instrução o salto somente será executado se o resultado for positivo, e na segunda, se for negativo.

As instruções deste subgrupo não são comumente usadas, porque requerem endereçamento absoluto e também porque um bit de sinal pode ser lido de diversas maneiras.

SUBGRUPO 7 - INSTRUÇÕES DE SALTO CONDICIONAL RELATIVAS AO ESTADO DA FLAG DE PARIDADE/EXCESSO *(PARITY/OVERFLOW)*

São duas instruções que permitem que seja efetuado um salto somente se a bandeira de paridade/excesso estiver nas condições especificadas pela instrução.

Esta flag é o bit 2 do registro F, e é uma bandeira de propósito duplo. Certas instruções usam-na para indicar "excesso", enquanto outras utilizam-na para armazenar o resultado de um teste de paridade.

O conceito de excesso, aqui não se aplica ao excesso binário, mas ao excesso de um complemento de dois aritmético ilustrado a seguir:

Consideremos:

ØA ADD 5C = 66

Em decimal: 1Ø ADD 92 = 1Ø2

Correto - não houve excesso

6A ADD 32 = 9C

Em decimal: 106 ADD 50 = - 100 (lembre-se do bit de sinal: 256-156=100)

Errado - houve excesso

Excessos também acontecem na subtração.

Veja:

83 SUB 14 = 6F

Em decimal: -125 SUB 20 = 111

Errado - houve excesso

A flag excesso/paridade é setada quando ocorre excessos. → Nº > 127 ou Nº < -128

Lápis e papel na mão para descobrir as operações exemplificadas acima.

O conceito de paridade refere-se ao número de bits setados em um determinado byte. A paridade existirá quando o número de bits setados for par.

Por exemplo:

O byte 01010101 tem paridade par e a flag setada;

O byte 00000111 tem paridade ímpar e a flag resetada

Resumindo:

1- Instruções ADD, ADC, SBC, CP, usando registros simples e instruções ADC e SBC usando pares de registros tem seu resultado testado em função do excesso.

2- Instruções AND, OR e XOR e rotações tem seu resultado testado em função da paridade.

3- Uma instrução INC vai setar esta flag se o resultado for 8Ø, e uma instrução DEC vai setar a flag se o resultado for 7F.

4- Várias outras instruções também afetam a flag de paridade/excesso.

As instruções deste subgrupo são:

MNEMÔNICA	CÓDIGO HEXA
JP PO, END	E2 END
JP PE, END	EA END

Na primeira instrução, o salto será executado se a paridade for ímpar ou não houver excesso.

Na segunda, o salto será executado se a paridade for par ou houver excesso.

As instruções deste subgrupo não são muito utilizadas, devido à confusão que podem causar e porque também podem ser substituídas por outras instruções.

CAPÍTULO 17 — INSTRUÇÃO DJNZ, E

Esta única instrução deste grupo é uma das mais úteis e das mais usadas de todo o conjunto de instruções do microprocessador Z8Ø.

MNEMÔNICA	CÓDIGO HEXA
DJNZ, e	1Ø e

A mnemônica significa "decremente o registro B e efetue um salto relativo se a flag zero estiver resetada".

Esta instrução pode ser comparada a um loop Basic, com passo negativo, do tipo:

```
FOR F= 1Ø TO 1 STEP -1:...IF..THEN....:NEXT
```

Nesse loop, a variável de controle F é inicializada em 1Ø, e a cada passagem pelo passo, ela é decrementada em uma unidade, até atingir o valor limite. Repare que a declaração IF...THEN... condicional equivale à condição da flag zero na instrução DJNZ, e.

A instrução "DJNZ, e" é usada de uma maneira muito similar. Primeiramente o programador deve especificar o tamanho da variável do loop e armazená-la no registro B; a seguir, as instruções que serão repetidas, e, finalmente, muito cuidado deve ser tomado, a fim de que o valor "e" seja apropriado (esteja contido na faixa limite).

CAPÍTULO 18 — INSTRUÇÕES DA PILHA (*STACK*)

Em muitos programas em linguagem de máquina, o uso extensivo da pilha da máquina é feito pelo programador como um lugar para guardar dados, e, pelo microprocessador, para guardar endereços que serão utilizados posteriormente. As instruções que formam este grupo podem ser subdivididas em dois subgrupos do usuário e três subgrupos do microprocessador.

SUBGRUPO 1 - INSTRUÇÕES PUSH E POP

Estas instruções permitem que o programador "PUSH", isto é, guarde, ou salve, dois bytes de dados na pilha da máquina, para mais tarde, "POP", ou seja, copie da pilha aqueles valores.

Este par de bytes pode ser copiado de e para um par de registros especificado.

MNEMÔNICA	CÓDIGO HEXA
PUSH AF	F5
PUSH HL	E5
PUSH BC	C5
PUSH DE	D5
PUSH IX	DDE5
PUSH IY	FDE5
POP AF	F1
POP HL	E1
POP BC	C1
POP DE	D1
POP IX	DDE1
POP IY	FDE1

Quando uma instrução PUSH é executada, o ponteiro da pilha (*stack pointer*) é primeiramente decrementado para apontar para uma locação livre.

Uma cópia do conteúdo do registro mais significativo do par de registros envolvido é então feita nesta locação. Então o ponteiro da pilha é decrementado novamente para armazenar na nova locação o valor contido no registro menos significativo do par envolvido.

As ações inversas a essas descritas são executadas no caso de instruções POP.

Note que sempre que uma dessas instruções é executada, ao final delas, o ponteiro da pilha vai apontar para aquela locação especificada pelo programador, ou aquela onde estava o processamento normal do programa.

NOTA IMPORTANTE:

Estas instruções são sempre usadas aos pares, ou seja, para cada instrução PUSH utilizada, obrigatoriamente deve haver uma instrução POP equivalente.

SUBGRUPO 2 - INSTRUÇÕES DE MUDANÇA DO VALOR DA PILHA

As instruções deste subgrupo não são muito usadas, mas, em algumas ocasiões são muito úteis.

MNEMÔNICA	CÓDIGO HEXA
EX (SP), HL	E3
EX (SP), IX	DDE3
EX (SP), IY	FDE3

Estas instruções permitem que o programador troque o valor corrente armazenado num par de registros especificado, pela última entrada na pilha da máquina, sem alterar o conteúdo do par de registros SP.

O uso dessas instruções é um tanto confuso, principalmente em se tratando de rotinas em linguagem de máquina mais extensas, e elas são mais bem consideradas como alternativas a PUSH e POP em casos especiais.

Considere, por exemplo, a situação a seguir:

Um valor qualquer XX está na pilha da máquina, enquanto que um valor qualquer YY está armazenado no par HL.

É desejo do programador trocar estes valores por outros.

Existem duas maneiras de fazê-lo:

1- Usando a instrução EX (SP), HL ou

2- Usando outro par de registros para armazenamento temporário para XX, da seguinte forma:

POP BC	Salva XX no par BC
PUSH HL	YY é colocado na pilha
PUSH BC	Move XX para HL de
POP HL	um modo ou de outro

As instruções desse subgrupo também podem ser utilizadas para manipular endereços de retorno.

SUBGRUPO 3 - INSTRUÇÕES CALL

As instruções em linguagem de máquina CALL são diretamente equivalentes ao comando Basic, GOSUB.

As instruções estão incluídas neste grupo porque o microprocessador usa a pilha da máquina como uma área onde os endereços de retorno são armazenados.

Existem nove instruções neste subgrupo que permitem que uma rotina seja chamada condicional ou incondicionalmente em relação ao estado das principais flags.

MNEMÔNICA	CÓDIGO HEXA	COMENTÁRIO
CALL END	CD END	Incondicional
CALL C, END	DC END	Flag C setada
CALL NC, END	D4 END	Flag C resetada
CALL Z, END	CC END	Flag Ø setada
CALL NZ, END	C4 END	Flag Ø resetada
CALL M, END	FC END	Flag sin. setada
CALL P, END	F4 END	Flag sin. reset.
CALL PE, END	EC END	Flag PE setada
CALL PO, END	E4 END	Flag PE resetada

As ações de uma instrução CALL são:

1- O valor corrente do PC, ou contador de programas, isto é, o endereço da primeira locação após "END" da instrução CALL, é salvo na pilha da máquina. O ponteiro da pilha é manipulado como numa instrução PUSH.

O byte mais significativo do contador de programas vai para a locação seguinte à do byte menos significativo.

2- O endereço é então copiado no contador de programas e a execução do programa propriamente dito continua.

SUBGRUPO 4 - INSTRUÇÕES RET

As instruções em código de máquina RET são diretamente equivalentes ao comando Basic RETURN.

Também são nove instruções neste subgrupo, que permitem o retorno condicional ou incondicional, dependendo do estado das flags principais.

MNEMÔNICA	CÓDIGO HEXA	COMENTÁRIOS
RET	C9	Incondicional
RET C	D8	Flag C setada
RET NC	DØ	Flag C resetada
RET Z	C8	Flag Ø setada
RET NZ	CØ	Flag Ø resetada
RET M	F8	Flag sin. setada
RET P	FØ	Flag sin. reset.
RET PE	E8	Flag PE setada
RET PO	EØ	Flag PE resetada

A ação de uma instrução RET é a de copiar a última entrada na pilha da máquina para o contador de programa. No entanto, o ponteiro da pilha é incrementado duas vezes.

Não é nada comum em Basic, manipular a pilha de GOSUB, mas em linguagem de máquina freqüentemente somos obrigados a fazê-lo, o que requer muitos cuidados com os valores a serem processados, bem como com os endereços a serem manipulados.

SUBGRUPO 5 - INSTRUÇÕES RST (*RESTART*)

O último subgrupo de instruções deste grupo contém as

instruções especiais RST, ou RESTART (REINÍCIO).

Estas instruções, muito úteis por sinal, são na verdade instruções CALL, que diferem daquelas apenas por não requererem a especificação de endereço, e obviamente, a economia de memória, já que estas utilizam apenas um byte.

MNEMÔNICA	CÓDIGO HEXA	COMENTÁRIOS
RST 0000	C7	CALL &H0000
RST 0008	CF	CALL &H0008
RST 0010	D7	CALL &H0010
RST 0018	DF	CALL &H0018
RST 0020	E7	CALL &H0020
RST 0028	EF	CALL &H0028
RST 0030	F7	CALL &H0030
RST 0038	FF	CALL &H0038

Detalhando estas instruções:

RST 0000 (*CHKRAM*)

Quando o micro computador é ligado, é neste endereço que começa a execução de qualquer processamento. O nome original desta subrotina da ROM BIOS é CHKRAM, abreviação de "check the RAM", ou seja, checar a memória RAM, e qualquer coisa que estiver armazenada nela estará perdida, bem como o conteúdo de qualquer um dos registros disponíveis do Z80.

RST 0008 (*SYNCHR*)

Esta rotina é utilizada pelo interpretador Basic para checar a sintaxe de erros. Se nenhum erro for encontrado, o processamento continua através de CHRGTR (0010). Não convém ser utilizada em rotinas de linguagem de máquina.

RST ØØ1Ø (*CHRGTR*)

O interpretador Basic chama este endereço para buscar o próximo caractere ou palavra chave de uma linha de programa Basic. Também não convém ser utilizada por programadores.

RST ØØ18 (*OUTDO*)

Esta instrução envia o conteúdo do acumulador para o periférico selecionado (discos, impressoras, terminais etc).

RST ØØ2Ø (*DCOMPR*)

Esta chamada simplesmente compara o conteúdo do par de registros HL com o conteúdo do par DE.

RST ØØ28 (*GETYPR*)

Aqui, o interpretador Basic procura qual tipo de número em ponto flutuante o acumulador está utilizando.

RST ØØ3Ø (*CALLF*)

Esta instrução desenvolve uma chamada para slot. O byte seguinte à instrução RST ØØ3Ø descreve o slot a ser empregado, e os dois bytes seguintes armazenam o endereço a ser chamado.

NOTA: O byte que descreve o slot tem o seguinte formato:

BIT Num.:	7	6	5	4	3	2	1	Ø
Conteúdo:	F	x	x	x	S	S	P	P

Onde:

PP (Ø A 3) – Número do slot primário a ser selecionado.

SS (Ø a 3) - Número do slot secundário.

F (Ø a 1) - Flag que será setada (valerá 1) se um slot secundário estiver em uso, e em caso contrário será resetada.

Os bits 4 a 6 não são utilizados.

RST ØØ38 (*KEYINT*)

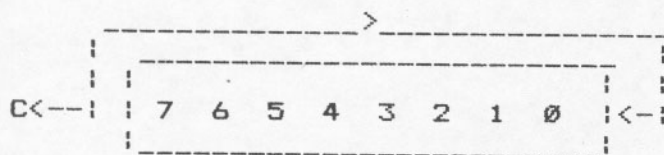
O sistema MSX opera no modo de interrupção 1, e é isto que a instrução RST ØØ38 executa quando ocorre uma interrupção mascarada. Estas interrupções ocorrem a cada Ø.Ø24 seg. e é efetuada então uma leitura de teclado, entre outras ações.

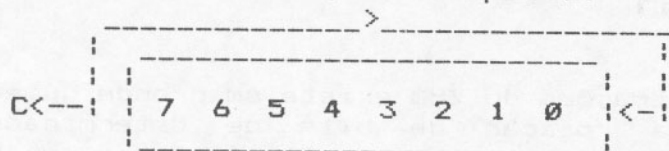
CAPÍTULO 19 — INSTRUÇÕES DE ROTAÇÃO

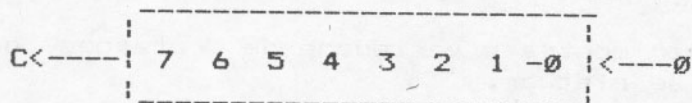
No conjunto de instruções do Z80 existe um grande número de instruções para rotação de bits de determinados bytes. São instruções quase sempre muito úteis (a Konami que o diga - quase sempre utiliza este tipo de instrução para "criptografar" mensagens em seus programas).

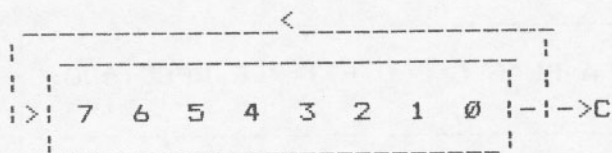
Rodar um byte para a esquerda tem o efeito de duplicar o seu valor, sem perder o conteúdo do bit mais significativo, enquanto que rodar o byte para a direita significa reduzir à metade seu valor.

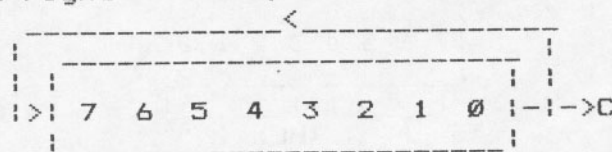
O diagrama a seguir mostra a variedade de rotações que são possíveis de se efetuar.

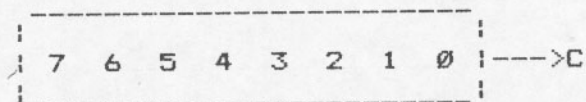
RLC e RLCA (*A de Acumulador*)
"Rotate left with carry" - Rotação à esquerda com transporte.

O BIT 7 VAI PARA A FLAG C E PARA O BIT Ø.

RL e RLA *(A de acumulador)*
"Rotate left" - Rotação à esquerda.

O BIT 7 VAI PARA A FLAG C QUE VAI PARA O BIT Ø.

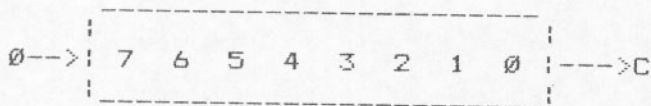
SLA
"Shift left" - Deslocamento à esquerda

O BIT Ø É RESETADO E O BIT 7 VAI PARA A FLAG C.

RRC e RRCA *(A de Acumulador)*
"Rotate right with carry" - Rotação à direita com transporte

O BIT Ø VAI PARA A FLAG C E PARA O BIT 7.

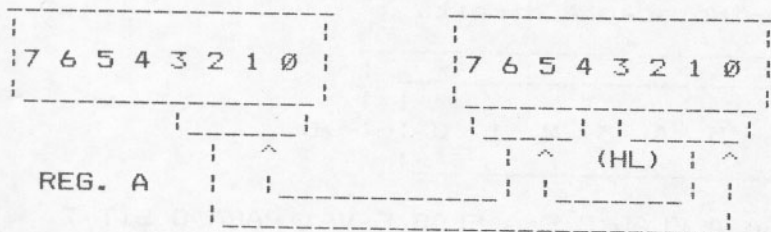
RR e RRA
"Rotate right" - Rotação à direita

O BIT Ø VAI PARA A FLAG C E A FLAG C VAI PARA O BIT 7.

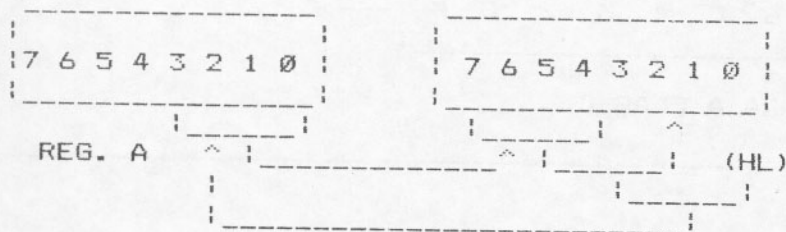
SRA A
"Shift right" - Deslocamento à direita

O BIT Ø VAI PARA A FLAG C.

SRL
"Logical shift right" - Deslocamento lógico à direita.

```
        ____________________________
       |                            |
  Ø--->| 7  6  5  4  3  2  1  Ø     |--->C
       |____________________________|
```

O BIT Ø VAI PARA A FLAG C E O BIT 7 É RESETADO.

RLD

"Rotate digit left and right between accumulator and location (HL)" - Rotação de dígitos à esquerda e à direita entre registro A e par HL.

RRD

"Rotate digit right and left between location (HL) and accumulator" - Rotação de dígitos à esquerda e à direita entre registro A e par HL.

A tabela abaixo mostra as instruções deste grupo:

Oper.	RLC	RL	SLA	RRC	RR	SRA	SRL
A	CBØ7	CB17	CB27	CBØF	CB1F	CB2F	CB3F
H	CBØ4	CB14	CB24	CBØC	CB1C	CB2C	CB3C
L	CBØ5	CB15	CB25	CBØD	CB1D	CB2D	CB3D
B	CBØØ	CB1Ø	CB2Ø	CBØ8	CB18	CB28	CB38
C	CBØ1	CB11	CB21	CBØ9	CB19	CB29	CB39
D	CBØ2	CB12	CB22	CBØA	CB1A	CB2A	CB3A
E	CBØ3	CB13	CB23	CBØB	CB1B	CB2B	CB3B
(HL)	CBØ6	CB16	CB26	CBØE	CB1E	CB2E	CB3E
(IX+D)	DDCB D Ø6	DDCB D 16	DDCB D 26	DDCB D ØE	DDCB D 1E	DDCB D 2E	DDCB D 3E
(IY+D)	FDCB D Ø6	FDCB D 16	FDCB D 26	FDCB D ØE	FDCB D 1E	FDCB D 2E	FDCB D 3E

Existem ainda quatro instruções de um byte, para rodar o acumulador, e duas instruções de manipulação de "nibbles", ou seja, quatro bits de um byte dividido ao meio, do bit Ø ao bit 3 e do bit 4 ao bit 7, respectivamente, nibble menos significativo e nibble mais significativo.

MNEMÔNICA	CÓDIGO HEXA
RLCA	Ø7
RLA	17
RRCA	ØF
RRA	1F

RRD	ED67
RLD	ED6F

Quanto às flags:

1- Todas as instruções, exceto RLD e RRD afetam a flag de transporte.

2- Todas as instruções, exceto as quatro instruções de byte simples, afetam respectivamente as flags zero, de sinal e de paridade/excesso.

CAPÍTULO 2Ø — INSTRUÇÕES DE MANIPULAÇÃO DE BITS

Estas instruções permitem ao programador testar, setar ou resetar qualquer bit de um determinado byte armazenado em um registro ou em um endereço da memória. Estes três tipos de instrução serão vistos a seguir, divididos em três subgrupos, conforme suas mnemônicas.

SUBGRUPO 1 - INSTRUÇÕES BIT

Estas instruções permitem que o programador determine o estado de um bit específico.

Uma instrução BIT seta a flag zero se o bit testado estiver resetado (valer Ø), e vice-versa.

SUBGRUPO 2 - INSTRUÇÕES SET

Estas instruções permitem ao programador setar determinado bit de um byte. Nenhuma flag é afetada.

SUBGRUPO 3 - INSTRUÇÕES RES (*RESET*)

Inversamente ao grupo anterior, estas instruções permitem que o programador resete um determinado bit de um byte.

Também não afetam nenhuma das flags.

As instruções destes três subgrupos estão na tabela a seguir:

		BIT Ø	BIT 1	BIT 2	BIT 3	BIT 4	BIT 5	BIT 6	BIT 7
REG.	BIT	47	4F	57	5F	67	6F	77	7F
A	RES	87	8F	97	9F	A7	AF	B7	BF
CB...	SET	C7	CF	D7	DF	E7	EF	F7	FF
REG.	BIT	44	4C	54	5C	64	6C	74	7C
H	RES	84	8C	94	9C	A4	AC	B4	BC
CB...	SET	C4	CC	D4	DC	E4	EC	F4	FC
REG.	BIT	45	4D	55	5D	65	6D	75	7D
L	RES	85	8D	95	9D	A5	AD	B5	BD
CB...	SET	C5	CD	D5	DD	E5	ED	F5	FD
REG.	BIT	4Ø	48	5Ø	58	6Ø	68	7Ø	78
B	RES	8Ø	88	9Ø	98	AØ	A8	BØ	B8
CB...	SET	CØ	C8	DØ	D8	EØ	E8	FØ	F8
REG.	BIT	41	49	51	59	61	69	71	79
C	RES	81	89	91	99	A1	A9	B1	B9
CB...	SET	C1	C9	D1	D9	E1	E9	F1	F9
REG.	BIT	42	4A	52	5A	62	6A	72	7A
D	RES	82	8A	92	9A	A2	AA	B2	BA
CB...	SET	C2	CA	D2	DA	E2	EA	F2	FA
REG.	BIT	43	4B	53	5B	63	6B	73	7B
E	RES	83	8B	93	9B	A3	AB	B3	BB
CB...	SET	C3	CB	D3	DB	E3	EB	F3	FB
(HL)	BIT	46	4E	56	5E	66	6E	76	7E
CB...	RES	86	8E	96	9E	A6	AE	B6	BE
	SET	C6	CE	D6	DE	E6	EE	F6	FE

CAPÍTULO 21 - INSTRUÇÕES DE MANIPULAÇÃO DE BLOCOS

São, no total, oito instruções de manipulação de blocos. Estas instruções são muito úteis e muito interessantes, pois permitem ao programador mover um bloco de dados de uma área da memória para outra, ou procurar uma área da memória.

Para mover um bloco de dados, o endereço-base deve estar armazenado no par de registros HL, o endereço de destino deve estar armazenado no par de registros DE, e o número de bytes do bloco, ou seja, o seu comprimento deve estar armazenado no par de registros BC.

Para procurar na memória a primeira ocorrência de um determinado valor, o endereço-base também deve estar armazenado no par HL, o número de bytes da área de pesquisa deve estar no par BC, e o registro A deve armazenar uma cópia do valor a ser encontrado.

As instruções deste grupo são:

MNEMÔNICA	CÓD. HEXA	COMENTÁRIOS
LDIR	EDBØ	Mover bloco-incrementa
LDDR	EDB8	Mover bloco-decrementa
CPIR	EDB1	Procura bloco-incrementa
CPDR	EDB9	Procura bloco-decrementa
LDI	EDAØ	Mover byte-incrementa
LDD	EDA8	Mover byte-decrementa
CPI	EDA1	Compara byte-incrementa
CPD	EDA9	Compara byte-decrementa

As primeiras quatro instruções são automáticas, mais rápidas, e mais utilizadas que as quatro últimas, não-automáticas.

Vamos analisar cada instrução detalhadamente.

INSTRUÇÕES AUTOMÁTICAS

1 - **LDIR**- *"Load location (DE) with location (HL), increment DE, HL, decrement BC and Repeat until BC=Ø"*. Esse é o nome em inglês. Traduzindo, LDIR vai mover um bloco de dados cujo endereço inicial está armazenado no par de registros HL, para uma área da memória, cujo endereço inicial está armazenado no par de registros DE (DEstino) e o comprimento desse bloco está armazenado no par de registros BC.

Quando em operação, o byte armazenado em HL é transferido para o par DE. O contador, que no caso é o par BC, é decrementado, e os valores armazenados em HL e DE são incrementados. Enquanto o par BC não chegar a Ø, o processo será repetido.

2 - **LDDR**- *"Load location (DE) with location (HL), decrement DE, HL and BC and Repeat until BC=Ø"*.

Repare nos títulos das duas instruções; enquanto a primeira incrementa DE e HL, esta decrementa estes dois pares de registros, juntamente com o contador de bytes, o par BC.

Portanto esta instrução requer como endereço base do bloco a sua última locação.

3- **CPIR**- "*Compare location (HL) and Accumulator, increment HL, decrement BC and Repeat until BC=Ø*".

Esta instrução procura em uma área específica da memória pela ocorrência de um determinado valor, pela primeira vez. O par HL deve armazenar o endereço base; o par BC deve armazenar o número de bytes a serem pesquisados e o registro A (acumulador) deve armazenar o valor a ser pesquisado.

Em operação, o byte armazenado em HL é comparado com o armazenado no registro A.

Se a comparação não for verdadeira, então a instrução decrementa o par BC e incrementa o par HL, para proceder à próxima comparação.

A operação continua até uma delas ter resultado verdadeiro, ou seja, o conteúdo do endereço apontado por HL ser igual ao conteúdo do registro A, ou então, se não ocorrer uma comparação verdadeira, a operação termina quando BC=Ø. Neste caso, as flags zero e de paridade/excesso serão resetadas.

4- **CPDR**- "*Compare location (HL) and Accumulator, decrement HL and BC and Repeat until BC=Ø*". A operação desenvolvida por esta instrução é similar à anterior, com a única diferença que o bloco de dados é pesquisado a partir do seu último endereço.

INSTRUÇÕES NÃO AUTOMÁTICAS

1- **LDI**- "*Load location (DE) with location (HL), increment DE, HL, decrement BC*". Foi por isso que nas instruções automáticas eu coloquei "Repeat until..." com letra maiúscula. A única diferença entre estas instruções e as automáticas é que neste grupo não há essa repetição automática.

A execução desta instrução faz com que um byte armazenado no endereço apontado por HL seja movido ou transferido para o endereço apontado por DE. O valor armazenado no par BC é decrementado. A flag de paridade/excesso será setada, a menos que o par BC atinja o valor Ø. Os valores nos pares DE e HL são incrementados.

2 - **LDD**- "*Load location (DE) with location (HL), decrement DE, HL and BC*". A única diferença entre esta instrução e a anterior é que esta decrementa os pares DE e HL, em vez de incrementá-los.

3 - **CPI**- "*Compare location (HL) and accumulator, increment HL and decrement BC*". A execução desta instrução vai fazer com que o byte endereçado por HL seja copiado no microprocessador e armazenado, enquanto o valor em HL é incrementado e o valor em BC é decrementado. O valor armazenado no microprocessador é então comparado com o valor do acumulador. Se o resultado da comparação for verdadeiro (igualdade), então a flag zero será setada, e, de outra forma, ela será resetada. A flag de sinal será resetada e a flag de paridade/excesso também será resetada, até que o valor em BC atinja Ø, quando ela será setada.

4 - **CPD**- "*Compare location (HL) and accumulator, decrement HL and BC*". Esta instrução é similar à anterior, com a exceção de que o valor armazenado no par HL é decrementado.

CAPÍTULO 22 — INSTRUÇÕES DE ENTRADA E SAÍDA *(INPUT/OUTPUT)*

Estas instruções de entrada/saída de dados (bytes) permitem que o programador receba dados de uma fonte externa (IN) ou envie dados para um dispositivo externo (OUT).

São instruções simples, não-automáticas e automáticas.

Em todos os casos de instruções de entrada/saída, os dados manipulados são bytes de 8 bits enviados em paralelo.

Quando o microprocessador está executando uma instrução IN, ele pega o byte envolvido na instrução, nas vias de dados (DATA BUS) e copia-o em um determinado registro. O pino de controle IORQ é ativado, bem como o pino RD, durante a sua execução.

Quando o Z80 está processando uma instrução OUT, ele coloca uma cópia do valor armazenado em um determinado

registro nas vias de dados, de onde ele será coletado por um dispositivo externo. Os pinos IORQ e WR se tornarão ativos durante sua execução.

Em adição ao estado dos pinos RD, WR e IORQ um dispositivo externo também será ativado pelo uso de um endereço apropriado colocado nas vias de endereçamento (ADDRESS BUS), durante a execução das instruções IN e OUT.

Este endereço é denominado "PORTA DE ENDEREÇO" que no caso do Z80 é um endereço de 16 bits.

As instruções deste grupo são:

MNEMÔNICA	CÓD. HEXA	I/O REG.	E.P. ALTO	E.P. BAIXO
IN A, (DD)	DB DD	A	A	DD
IN A, (C)	ED78	A	B	C
IN H, (C)	ED60	H	B	C
IN L, (C)	ED68	L	B	C
IN B, (C)	ED40	B	B	C
IN C, (C)	ED48	C	B	C
IN D, (C)	ED50	D	B	C
IN E, (C)	ED58	E	B	C
OUT (DD), A	D3 DD	A	A	DD
OUT (C), A	ED79	A	B	C
OUT (C), H	ED61	H	B	C
OUT (C), L	ED69	L	B	C
OUT (C), B	ED41	B	B	C
OUT (C), C	ED49	C	B	C
OUT (C), D	ED51	D	B	C
OUT (C), E	ED59	E	B	C

Nessa listagem, **E.P. ALTO** quer dizer byte mais significativo do endereço da porta, e **E.P. BAIXO** quer dizer byte menos significativo do endereço da porta.

As instruções automáticas e não automáticas são:

MNEMÔNICAS	CÓD. HEXA	COMENTÁRIOS
INI	EDA2	Não automática-Incrementa
INIR	EDB2	Automática-Incrementa
IND	EDAA	Não automática-Decrementa
INDR	EDBA	Automática-Decrementa
OUTI	EDA3	Não automática-Incrementa
OUTIR	EDB3	Automática-Incrementa
OUTD	EDAB	Não automática-Decrementa
OUTDR	EDBB	Automática-Decrementa

As mnemônicas querem dizer:

- **INI**- Carrega locação apontada pelo par HL, com entrada da porta (C), incrementa HL e decrementa B.

- **INIR**- Carrega locação apontada por HL, com entrada da porta (C), incrementa HL e decrementa B, até que B=Ø.

- **IND**- Carrega locação apontada por HL, com entrada da porta (C), decrementa HL e B.

- **INDR**- Carrega locação apontada por HL, com entrada da porta (C), decrementa HL e B até que B=Ø.

- **OUTI**- Carrega porta de saída (C) com locação apontada por (HL), incrementa HL e decrementa B.

- **OUTIR**- Carrega porta de saída (C) com locação apontada por (HL), incrementa HL e decrementa B, até que B=Ø.

- **OUTD**- Carrega porta de saída (C) com locação apontada por (HL), decrementa HL e B.

- **OUTDR**- Carrega porta de saída (C) com locação apontada por (HL), decrementa HL e B, e repete até que B=Ø.

CAPÍTULO 23 — INSTRUÇÕES DE INTERRUPÇÕES

Existem ao todo sete instruções que permitem que o programador manipule o sistema de interrupções do Z80. São elas:

MNEMÔNICA	CÓDIGO HEXA
EI	FB
DI	F3
IM 0	ED46
IM 1	ED56
IM 2	ED5E
RETI	ED4D
RETN	ED45

Vamos analisar cada uma dessas instruções:

- **EI** (*Enable Interrupt* — Permite Interrupções) — Quando ligamos o microcomputador, um sistema de interrupção "mascarada" é habilitado para interromper o funcionamento do Z80. Em outros termos, quando ligamos o micro, o seu microprocessador imediatamente começa a

trabalhar, executando as rotinas da ROM BIOS. Nessas rotinas, necessariamente deve haver um sistema de interrupção do microprocessador, para que ele possa executar um reconhecimento do teclado, por exemplo, no sentido de reconhecer se alguma tecla foi pressionada.

- **DI** (*Disable Interrupt*- Desabilita Interrupções) -
Em qualquer ponto de qualquer rotina em linguagem de máquina, o programador pode decidir "desligar" o sistema de interrupção mascarada, através dessa instrução DI, o que torna o microprocessador insensível a qualquer sinal do pino INT. Através da utilização desta instrução, em alguns casos, chegamos a ganhar mais de 50 % de tempo de processamento, já que não existe mais a leitura de teclado.

-**IM 0**- São três modos de interrupção. Este modo é selecionado automaticamente pelo microprocessador quando ligamos o microcomputador pela primeira vez, ou também pela execução desta própria instrução. Mas este modo não é utilizado pelo padrão MSX.

- **IM 1** - Este modo de interrupção é selecionado somente pela execução desta instrução, e é o modo utilizado pelo padrão MSX. O sistema operacional contido nos primeiros 16K da ROM do micro possui esta instrução como parte da rotina de inicialização.

Neste modo, a instrução RST &H0038 será sempre selecionada após receber um sinal do pino INT, o que significa que o sistema de interrupção mascarado foi habilitado. No padrão MSX, a rotina em linguagem de máquina com início em &H0038 atualiza o relógio de tempo real e faz a leitura do teclado.

- **IM 2**- Este modo não é utilizado pelo padrão MSX, mas, dos três modos de interrupção possíveis é o mais poderoso. Neste modo, um dispositivo periférico pode indicar ao microprocessador qual das subrotinas deve ser executada após receber a interrupção mascarada. O conteúdo do registro I e o byte fornecido pelo

dispositivo periférico são usados juntos para formar um endereço de 16 bits, utilizado para endereçar uma tabela de vetores, previamente preparada na memória.

- **RETI**- Esta instrução é um "retorno" especial, para ser empregado em rotinas de interrupção mascarada. O efeito desta instrução é o de retornar com a mesma interrupção mascarada depois de ter sido interrompido o processamento normal.

- **RETN**- Esta instrução é similar à anterior, mas é aplicada no fim de uma rotina de interrupção não mascarada.

CAPÍTULO 24 — INSTRUÇÕES DIVERSAS

Existem ainda seis instruções que não foram mencionadas. São elas:

MNEMÔNICA	CÓDIGO HEXA
CPL	2F
NEG	ED44
SCF	37
CCF	3F
HALT	76
DAA	27

O significado das mnemônicas:

CPL- Complementa acumulador

NEG- Negativo do acumulador (complemento de 2)

SCF- Seta flag de transporte

CCF- Complementa flag de transporte

HALT- Aguarda uma interrupção ou um reset

DAA- Ajuste decimal do acumulador

Analisando:

- **CPL**- Esta é uma instrução simples que complementa o registro A, ou seja, seta o bit que está resetado e vice-versa. Esta operação é chamada de complemento de 1. Não afeta nenhuma flag.

- **NEG**- Esta instrução calcula o complemento de 2 do acumulador. As flags de sinal e zero dependem do resultado para serem alteradas. A flag de transporte será resetada se o valor original for zero; de outra forma,será setada, e a bandeira de paridade/excesso será setada se o valor original for &H8Ø.

- **SCF**- Seta a flag de transporte.

- **CCF**- Complementa a flag indicadora de transporte.

- **HALT**- Esta é uma instrução especial que faz com que o microprocessador pare o seu trabalho até que ocorra uma interrupção. No MSX, as únicas interrupções que podem ocorrer são as mascaradas.

- **DAA**- Esta é a instrução que faz o ajuste decimal do acumulador. Em aritmética binária BCD (*Binary Coded Arithmetic*), os algarismos de Ø a 9 são representados pelos "nibbles binários" ØØØØ a 1ØØ1, e os nibbles 1Ø1Ø a 1111 não são utilizados.

*OBS: SÓ FUNCIONA COM NUMEROS ENTRE 0 E 15 (0 E F H)

Por exemplo:

- o byte ØØØØØØØØ representa o número Ø

- o byte ØØ11 1ØØ1 representa o número 39

3 9

Portanto, esta instrução converte bytes em sua forma binária absoluta para a forma BCD.
A flag indicadora de sinal e a flag zero são afetadas pelo resultado, e a flag de paridade/excesso será setada se houver paridade par. O efeito na flag de transporte vai depender se houve excesso nas adições ou subtrações na forma BCD.

E assim nós terminamos esses capítulos extensos, complicados e chatos sobre todo o conjunto de instruções do microprocessador Z8Ø. Não é à toa que ele é considerado o mais complicado dos microprocessadores de 8 bits.

Os japoneses conseguiram demonstrar que um microprocessador de 8 bits é muito poderoso, e, nesse aspecto, a relação final custo/benefício, comparativamente a um micro de 16 bits, é por demais vantajosa.

Por exemplo, o **HITBIT**, da Sony, um MSX 2, que possui 8 canais de som, 128K de VRAM e 256 K de RAM!!! Pelo visto, dá para o começo. A sua resolução gráfica é idêntica à de um PC, sem contar a geração de cores simultâneas no vídeo.

Uma nova geração de micros está surgindo, através do **APPLE GS** (*Graphics & Sound*), que vem com apenas 32 canais de som, e uma resolução gráfica e de cor nunca antes vista em um micro desse porte, além de, em sua configuração básica, ser compatível com qualquer micro **APPLE** e **MCINTOSH**, e, através de uma placa de expansão, torna-se compatível com **PC XT**.

Não vale a pena citarmos a nova geração **PC-386** que está surgindo na matriz. São microcomputadores poderosíssimos, com desempenho semelhante à minicomputadores, mas com preço de micro. Refiro-me particularmente ao fantástico **COMPAQ III-386** que já

vem com 1M de RAM na placa, podendo ser expandido até 6.6M, 2 saídas paralelas, 2 duas seriais etc.

Não é preciso dizer mais nada - fiquemos com o sistema operacional do nosso MSX tropical, a partir do capítulo seguinte.

Estude muito bem todas as instruções e pratique bastante. Comece com rotinas simples, como, por exemplo, a transferência de um bloco de bytes armazenado na RAM, para a VRAM, ou seja, uma tela guardada na memória.

Se você quiser "piratear" ou traduzir aquele seu programa predileto, não pense que vai ser muito fácil.

Programas muito procurados não possuem quase nenhuma mensagem em caracteres ASCII, mas sim todas "criptografadas", ou codificadas, através de instruções, por exemplo, de adição, de rotação de bits, de comparações lógicas etc. Mas, apenas um detalhe: por exemplo, na palavra MSX, a relação entre os códigos de seus caracteres será sempre a mesma, ou seja, os códigos 4D, 53 e 58 representam os caracteres ASCII "MSX", que se tiverem seus bits rodados, ou se forem somados a um valor constante, ou se forem comparados logicamente a outro valor constante, a sua relação de valores será sempre a mesma. E isso você descobre facilmente através de um programa Basic.

O mesmo se aplica aos "SPRITES", que também podem ser descobertos através de programas Basic que mostrem no vídeo todos os sprites montados.

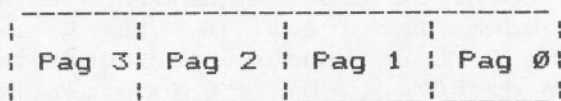
Experimente !

CAPÍTULO 25 — A PPI *(PROGRAMMABLE PERIPHERAL INTERFACE* OU INTERFACE PERIFÉRICA PROGRAMÁVEL)

O chip 8255 (PPI) é uma interface paralela para propósitos gerais, configurada como sendo três portas de dados de 8 bits, chamadas portas A, B e C, e uma porta de modo.

Este chip se comunica com o Z80 através de 4 portas de entrada/saída, por onde o teclado, a paginação da memória, o motor do cassete, a saída do cassete, o LED indicativo de CAPS LOCK (maiúsculas) e o click de tecla pressionada (sinal sonoro de pressão de uma tecla), podem ser controlados.

Para que a PPI inicialize o acesso a um periférico, basta apenas escrever ou ler na/da porta respectiva.

PORTA A DA PPI (Porta de entrada/saída &HA8)

Registro primário de slots

Esta porta de saída, conhecida como registro primário de slots, na terminologia MSX, é utilizada para controlar a paginação da memória. Sabemos que o Z8Ø pode acessar diretamente somente 64 k de memória.

O sistema MSX pode ter diversos dispositivos de memória, que se utilizam da mesma área, nas mesmas posições, e o microprocessador pode "paginar" a memória para qualquer desses dispositivos, conforme a necessidade.

O espaço de endereçamento do Z8Ø é visto como sendo uma duplicação para os lados, de 4 áreas de 64 k separadas, chamadas de Slots primários, numeradas de Ø a 3, distintos entre si por um sinal a partir dos pinos do Z8Ø.

O conteúdo do registro primário de slots determina qual sinal está sendo recebido, para então selecionar aquele respectivo slot.

Para maior flexibilidade, cada "página" de 16 k da área de endereçamento do Z8Ø pode ser selecionada de slots primários diferentes.

No esquema acima, você pode notar que são necessários dois bits do registro primário de slot para definir o número do slot primário para cada página.

O sistema MSX pode ter 4 slots primários, sendo o de número Ø aquele que contém a ROM BASIC, e por isso é chamado de slot do sistema. Cada um desses slots pode ser expandido para 4 slots de expansão.

Portanto, o número total de slots expandidos é de 16 !!! Estes 16 slots podem ter cada um 16 K de RAM, totalizando cerca de 1 Mb de espaço de memória, sendo este valor o máximo de RAM que um MSX pode gerenciar.

Mas, lembre-se de que você não pode acessar toda essa memória a partir do Basic. Você vai precisar ou do MSX DOS (*Disk Operating System* - Sistema Operacional de Discos) ou programas em linguagem de máquina para acessar memórias maiores que 64 k.

CONFIGURAÇÃO BÁSICA DOS SLOTS

```
        SLOT Ø    SLOT 1    SLOT 2    SLOT 3
                                              &HFFFF
        _____     _____     _____     _____
       |     |   |     |   |     |   |     |
       |Pag 3|   |Pag 3|   |Pag 3|   |Pag 3|    16K
       |     |   |     |   |     |   |     |
       |_____|   |_____|   |_____|   |_____|  &HCØØØ
        _____     _____     _____     _____   &HCØØØ
       |     |   |     |   |     |   |     |
       |Pag 2|   |Pag 2|   |Pag 2|   |Pag 2|    16K
       |     |   |     |   |     |   |     |
       |_____|   |_____|   |_____|   |_____|  &H8ØØØ
        _____     _____     _____     _____   &H7FFF
       |     |   |     |   |     |   |     |
       |Pag 1|   |Pag 1|   |Pag 1|   |Pag 1|    16K
       |     |   |     |   |     |   |     |
       |_____|   |_____|   |_____|   |_____|  &H4ØØØ
        _____     _____     _____     _____   &H3FFF
       |     |   |     |   |     |   |     |
       |Pag Ø|   |Pag Ø|   |Pag Ø|   |Pag Ø|    16K
       |     |   |     |   |     |   |     |
       |_____|   |_____|   |_____|   |_____|  &HØØØØ
```

Assim que você liga o micro, a primeira operação

desenvolvida pela sua ROM é saber em qual slot está localizada a sua RAM, nas páginas 2 e 3 (&H8000 até &HFFFF). O registro de slot primário é então "setado" para aquele slot, fazendo com que toda a área de RAM esteja disponível para você.

Vale lembrar que a ROM do seu MSX sempre estará localizada no slot primário 0, pois é este o slot selecionado quando o micro é ligado.

Outros dispositivos de memória podem ser colocados em qualquer slot.

Como vimos acima, a memória do sistema pode ser incrementada para até 16 áreas de 64 Kb, que quando existem, são feitas através de interfaces apropriadas, as famosas "Expansões de RAM"...

Uma interface de expansão conectada em algum slot primário pode proporcionar 4 slots secundários de 64 K cada, numerados de 0 a 3. Cada expansão possui, em seu hardware próprio, um registro denominado registro Secundário de slot, utilizado para selecionar qual slot secundário deve "aparecer" no primário.

As páginas também podem ser selecionadas de diferentes slots secundários. A paginação da memória obviamente é uma operação que requer certas cauções, particularmente quando outros mecanismos devem controlar uma expansão, ou a alteração de um página onde esteja sendo executado um programa, para outra qualquer, de qualquer outro slot, ainda mais se aquela que está sendo utilizada contém a pilha da máquina.

Existem algumas rotinas disponíveis ao usuário na ROM BIOS que podem simplificar o processo. O interpretador Basic possui 4 métodos para acessar extensões da ROM. Os 3 primeiros são para uso com rotinas em linguagem de máquina da ROM, da página 1 (&H4000 a &H7FFF). São eles:

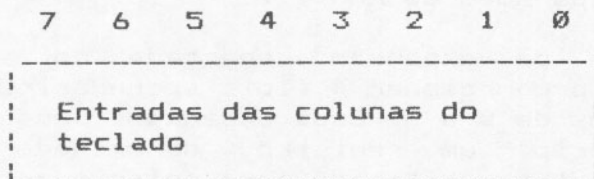
1- **Hooks** (Ganchos)

2- Declarações CALL

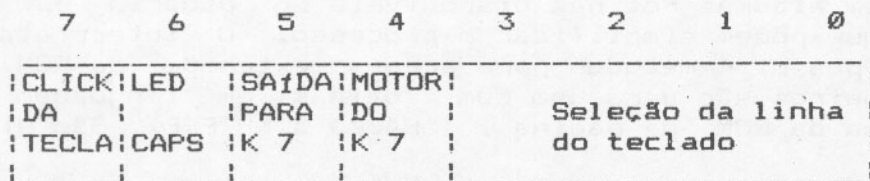
3- Nomes adicionais para dispositivos

O Interpretador Basic também pode executar um programa em linguagem Basic detectado na página 2 (&H8000 a &HBFFF), quando a ROM procura pela RAM, assim que o micro é ligado. O que o interpretador Basic não faz é utilizar alguma RAM "escondida" em outros dispositivos de memória.

PORTA B DA PPI (Porta de entrada/saída &HA9).

```
      7   6   5   4   3   2   1   0
    ________________________________
   :                                :
   :  Entradas das colunas do       :
   :  teclado                       :
   :________________________________:
```

Esta porta de entrada é utilizada para ler os 8 bits dos dados da coluna da linha selecionada do teclado. A conversão de um sinal recebido por uma pressão em uma tecla em códigos de caracteres é feita pelas interrupções da ROM, explicadas no capítulo respectivo.

PORTA C DA PPI (Porta de entrada/saída &HAB). &HAA

```
     7      6      5      4      3      2      1      0
   ____________________________________________________
  :CLICK:LED   :SAíDA:MOTOR:                           :
  :DA   :      :PARA :DO   :     Seleção da linha      :
  :TECLA:CAPS  :K 7  :K 7  :     do teclado            :
  :_____:______:_____:_____:___________________________:
```

Esta porta de saída controla diversas funções. Os 4 bits de seleção de linha de teclado indicam qual das 11 linhas de teclado, numeradas de Ø a 1Ø, devem ser lidas pela porta B da PPI.

O bit do motor do cassete determina o estado deste.
Ø = on
1 = off

O bit de saída para cassete é filtrado e atenuado antes de ser reconhecido no soquete DIN do cassete, como um sinal MIC.

Toda a geração de tons do cassete é desenvolvida por software.

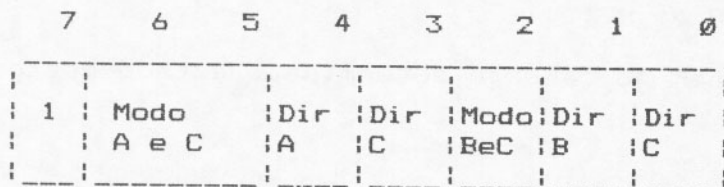
O bit de CAPS LOCK determina o estado do LED respectivo.
Ø = on
1= off

O click de tecla (sinal de saída) é atenuado e misturado com a saída de audio para a PSG (gerador de sons programável).

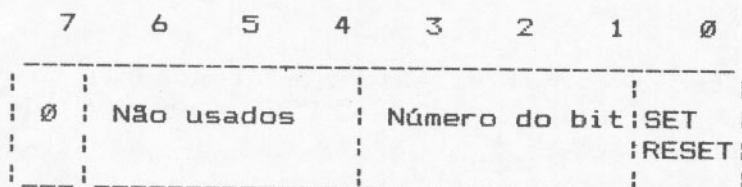
Existem rotinas padrão na ROM BIOS para acessar todas estas funções, que estão disponíveis ao usuário. Deve-se dar preferência à utilização das funções, sempre que possível, em vez de uma manipulação direta do hardware.

PORTA DE MODO DA PPI (Porta de entrada/saída &HAB).

Esta porta é utilizada para ajustar o modo de operação da PPI. Como o hardware do MSX foi desenvolvido para trabalhar em configurações particulares, apenas esta porta não pode ser alterada sob qualquer circunstância.

```
   7    6    5    4    3    2    1    Ø
 ____________________________________________
 |   |         |    |    |    |    |    |
 | 1 | Modo    |Dir |Dir |Modo|Dir |Dir |
 |   | A e C   |A   |C   |BeC |B   |C   |
 |___|_________|____|____|____|____|____|
```

Seleção de modo da PPI

O bit 7 deve ser 1 para se alterar o modo da PPI. Quando for Ø, a PPI desenvolve apenas uma função de SET/RESEt bits conforme esquema abaixo:

```
   7    6    5    4    3    2    1    Ø
 ____________________________________________
 |   |              |              |     |
 | Ø | Não usados   | Número do bit|SET  |
 |   |              |              |RESET|
 |___|______________|______________|_____|
```

Os bits do modo A e C determinam o modo de operação da porta A e os 4 bits superiores da porta C:

ØØ = modo normal (MSX)
Ø1 = modo expandido
1Ø = modo bidirecional

O modo DIR A determina a direção da porta A:

Ø = Saída (MSX)
1 = Entrada

O bit 3 do modo DIR C determina a direção dos 4 bits superiores da porta C:

Ø = Saída
1 = Entrada

O modo B e C determina o modo de operação da porta B e dos 4 bits inferiores da porta C:

Ø = modo normal
1 = modo expandido

O bit DIR B determina a direção da porta B:

Ø = Saída
1 = Entrada

O bit DIR C determina a direção apenas dos 4 bits inferiores da porta C:

Ø = Saída
1 = Entrada

A PORTA DE MODO DA PPI pode ser utilizada para "setar" ou "resetar" diretamente qualquer bit da porta C, quando o bit 7 vale Ø. Os números dos bits, de Ø a 7, determinam qual bit está sendo afetado naquele instante, sendo que seu novo valor é determinado pelo BIT SET/RESET. A grande vantagem deste modo é que uma simples saída pode ser facilmente modificada.

Como exemplo, o LED de CAPS LOCK pode ser ligado através da declaração Basic OUT &HAB,12 e desligado através de OUT &HAB,13.

CAPÍTULO 26 - SELEÇÃO DE SLOTS E SUAS VARIÁVEIS DE SISTEMA

COMO SELECIONAR E HABILITAR UM SLOT

Para selecionar páginas de slots por software, utiliza-se a porta de saída &HA8 correspondente à porta A da PPI.

O valor a ser enviado obviamente deve ser um número de 8 bits.

No capítulo anterior, você viu como manipular os bits do chamado Registro primário de slots.

Pois bem, digamos que você queira usar as páginas Ø, 1 e 3 do slot Ø e a página 2 do slot 1.

BITS	7	6	5	4	3	2	1	Ø
	Ø	Ø	Ø	1	Ø	Ø	Ø	Ø

Você deve portanto enviar para o endereço (porta) de entrada/saída, o valor &H16 (OUT &HA8,&H16).

Entretanto um método mais prático é o de se utilizar a chamada da ROM BIOS ENASLT (*Enable slot* – &HØØ24), a partir da linguagem de máquina.

SOFTWARES EM CARTUCHOS DE EPROM

Procedimentos de busca da ROM

Após selecionar a RAM disponível, o MSX procura então por cartuchos de EPROM, nos endereços de &H4ØØØ até &HBFFF, páginas 1 e 2. Ele procura por uma informação no início de cada página, do slot Ø ao slot 3, e, quando houver, nas expansões de slots.

A área de informação está sempre localizada no início de cada página, e deve ter um dos formatos abaixo:

```
TOPO DA   ___________
PÁGINA   |           |
         |     AB    | Os códigos de "AB" in-
 + &HØØ  |           | dicam que existe um
         |___________| cartucho de EPROM aqui
          ___________
         |           | Aqui está armazenado o
 + &HØ2  |   INICIO  | endereço de iniciali -
         |___________| zação daquela EPROM.
          ___________
         |           | Armazena o endereço da
 + &HØ4  |DECLARACÃO| rotina de expansão.
         |___________|
          ___________
         |           | Armazena o endereço da
 + &HØ6  |DISPOSIT.  | rotina de manipulação
         |___________| da expansão do dispos.
          ___________
         |           | Armazena o endereço de
 + &HØ8  |  TEXTO    | início do programa em
         |___________| Basic no cartucho.
          ___________
 + &HØA  |           |
         |           |
         |           |
         | Reservada |
         |           |
         |           |
         |           |
 + &H1Ø  |___________|
```

NOTA: AB, INIC., DECLARACÃO, DISPOSITIVO E TEXTO conterão zeros se não houver cartucho.

O Basic MSX executa os seguintes procedimentos para a procura de cartuchos de EPROM:

1- Checa a área de informação e encontra o tipo de rotina que está ali armazenada. Passa essa informação para a área de trabalho.

2- Se houver rotina de inicialização, executa-a.

3- Se houver programa em Basic no cartucho, executa-o.

Vamos analisar mais detalhadamente cada item.

Rotina de inicialização

Aqui está armazenado o endereço de inicialização específica daquele cartucho, na forma byte mais significativo e byte menos significativo (por exemplo, endereço &H4081, está armazenado como 81 e 40).

Essa rotina de inicialização pode alterar todos os registros do Z80, exceto o apontador (ponteiro) da pilha (SP). Ela retorna ao Basic através da instrução RET do Z80.

Note porém que esta rotina não precisa necessariamente ser uma rotina de inicialização. Ela pode muito bem ser um programa em linguagem de máquina a ser executado imediatamente após o micro ser ligado - por exemplo, um jogo.

Texto - Programa em linguagem Basic

Um cartucho de EPROM não precisa necessariamente ser um programa em linguagem de máquina. Pode conter um programa escrito em linguagem Basic.

Nesta área está armazenado o endereço inicial do programa Basic contido naquele cartucho.

Quando programando um software de cartucho em linguagem Basic, alguns cuidados devem ser tomados:

1- Quando houver mais de um cartucho conectado ao micro, este executará aquele que estiver no slot de número menor.

2- O cartucho deve estar localizado na página 2, endereços &H8ØØØ a &HBFFF, significando que a memória máxima para cartuchos em Basic é de 16K.

3- A RAM localizada na página 2 de qualquer slot é desabilitada e não pode ser utilizada pelo programa Basic do cartucho.

4- Para uma execução mais rápida do programa, as linhas de programa que contém GOTO e GOSUB devem ser alteradas para ponteiros.

Declaração: Rotina de declaração expandida

Utilizando-se a declaração do MSX Basic "CALL" você pode usar declarações expandidas que não estejam contidas na ROM do MSX. Um cartucho que contenha uma declaração expandida em Basic deve conter o endereço inicial da primeira rotina de declaração expandida na DECLARACÃO (&HØ4 a &HØ6). O cartucho deve estar localizado na página 1 de qualquer slot, exceto o slot do sistema, onde reside o Basic.

A sintaxe para declarações expandidas é:

CALL <nome da declaração>

CALL <nome da declaração> (argumento)

CALL pode ser abreviado pelo caractere "_".

Por exemplo : CALL FORMAT, para formatar discos, ou

-FORMAT.

Quando o Basic encontra a declaração CALL, armazena o nome da declaração na variável de sistema PROCNM, localizada a partir do endereço &HFD89, com 16 bytes de comprimento. O nome da declaração termina com Ø quando está armazenada em PROCNM, e por isso só pode ter 15 caracteres de comprimento.

Após a execução dessa declaração, o apontador do texto, o par de registros HL, apontará para o endereço após o nome na área DECLARACÃO.

O procedimento DECLARACÃO no cartucho então checará o conteúdo de PROCNM, e executará a rotina que corresponde ao nome da declaração.

Após sua execução, a flag de transporte será limpa e o apontador do texto (HL) passará a apontar para a nova posição da próxima declaração.

Se não houver declaração expandida, então a flag de transporte e o apontador (HL) não serão afetados, e haverá um retorno ao Basic com uma mensagem de "Erro de sintaxe".

Dispositivos - Rotinas de manipulação de dispositivos de expansão.

O Basic MSX possui a habilidade de conectar um dispositivo de expansão de entrada/saída para um cartucho de slot. A área no início do cartucho DISPOSIT. contém o endereço inicial da rotina de manipulação do dispositivo.

Alguns lembretes sobre essas rotinas sobre dispositivos:

1- O cartucho deve estar localizado na Página 1, de &H4ØØØ a &HBFFF.

2- Um cartucho (16K) pode ter até 4 dispositivos lógicos conectados.

3- O nome do dispositivo é armazenado na variável de sistema PROCNM, localizada a partir de &HFD89. Valem aqui as notas sobre nomes de declarações.

4- Quando o Basic encontra o nome do dispositivo, numa declaração OPEN, ou outra, que não seja conhecido pela ROM do MSX, então armazena no registro A o valor &HFF. Se não houver manipulação que corresponda ao nome do dispositivo, a flag de transporte será setada. Se houver o dispositivo, então a área de informação inicial (de Ø a 3) terá seu conteúdo transferido para o registro A, e a flag de transporte será resetada.

5- Valores armazenados no registro A a partir da variável de sistema DEVICE, quando existirem operações reais de entrada /saída:

Ø = OPEN
2 = CLOSE
4 = Entrada/saída randômica
6 = Saída seqüencial
8 = Entrada seqüencial
1Ø = Função LOC
12 = Função LOF
14 = Função EOF
16 = Função FPOS
18 = Caractere de "back-up"

DESCRIÇÃO DAS VARIÁVEIS DE SISTEMA CORRESPONDENTES AOS MECANISMOS DE SLOTS

Estado de cada slot

EXPTBL - indica qual slot está expandido.

Locação &HFCC1 – 4 bytes de comprimento:

```
EXPTBL  &HFCC1 - para slot Ø
        &HFCC2 - para slot 1
        &HFCC3 - para slot 2
        &HFCC4 - para slot 3
```

&H8Ø indica slot expandido
&HØØ indica sem expansão

SLTTBL – indica qual valor deve ser enviado ao registro para seleção de slots, válido somente se EXPTBL contiver &H8Ø.

Locação &HFCC5 – 4 bytes de comprimento:

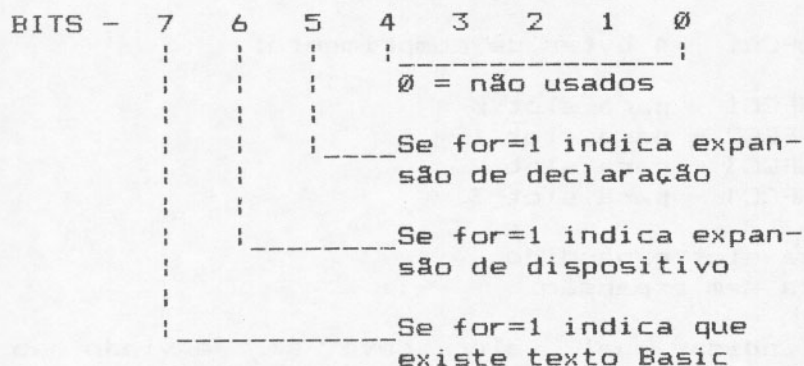
```
SLTTBL  &HFCC5 - para slot Ø
        &HFCC6 - para slot 1
        &HFCC7 - para slot 2
        &HFCC8 - para slot 3
```

Estado de cada página

SLTATR – armazena o número da página

Locação &HFCC9 – 64 bytes de comprimento:

```
SLTATR  &HFCC9 - slot básico Ø - expansão
                    do slot Ø - página Ø
        &HFCCA - slot básico Ø - expansão
                    do slot Ø - página 1
           ...
           ...
           ...
        &HFDØ7 - slot básico 3 - expansão
                    do slot 3 - página 2
        &HFDØ8 - slot básico 3 - expansão
                    do slot 3 - página 3
```

```
BITS -  7   6   5   4   3   2   1   Ø
        :   :   :   :_____________:
        :   :   :   Ø = não usados
        :   :   :
        :   :   :____Se for=1 indica expan-
        :   :        são de declaração
        :   :
        :   :________Se for=1 indica expan-
        :            são de dispositivo
        :
        :____________Se for=1 indica que
                     existe texto Basic
```

SLTWRK- Área de trabalho para cada página.

São permitidos 2 bytes por página.

Locação &HFDØ9 - 128 bytes de comprimento
2 bytes por página

```
SLTWRK  &HFDØ9 - slot básico Ø - expansão
                    do slot Ø - página Ø
        &HFDØA - slot básico Ø - expansão
                    do slot Ø - página Ø
        &HFDØB - slot básico Ø - expansão
                    do slot Ø - página 1
        &HFDØC - slot básico Ø - expansão
                    do slot Ø - página 1
            ...
            ...
            ...
        &HFD85 - slot básico 3 - expansão
                    do slot 3 - página 2
        &HFD86 - slot básico 3 - expansão
                    do slot 3 - página 2
        &HFD87 - slot básico 3 - expansão
                    do slot 3 - página 3
        &HFD88 - slot básico 3 - expansão
                    do slot 3 - página 3
```

CAPÍTULO 27 – O VDP (*VIDEO DISPLAY PROCESSOR* OU PROCESSADOR DE VÍDEO)

O chip 9128 VDP contém todos os circuitos eletrônicos necessários para gerar o sinal de vídeo.

Ele aparece para o Z80 como sendo duas portas de entrada/saída, chamadas de Porta de Dados e Porta de Comando.

Embora o VDP tenha seus próprios 16K de VRAM (Video RAM - Ram de vídeo), cujo conteúdo define a imagem da tela, ela (a memória) não pode ser acessada diretamente pelo Z 80. Apesar de utilizar duas portas de entrada/saída para modificar a VRAM, faz-se necessário ajustar várias condições de operação do VDP.

PORTA DE DADOS (Porta de entrada/saída &H98).

A porta de dados é usada para ler ou escrever bytes

simples na VRAM.

O VDP possui um registro de endereçamento interno apontando para uma locação na VRAM. Lendo a porta de dados, um byte será aceito a partir de uma locação da VRAM, enquanto escrevendo nessa porta fará com que um byte seja armazenado lá.

Após uma operação de leitura/escrita o registro de endereçamento será automaticamente incrementado para apontar para a próxima locação na VRAM. Uma seqüencia de bytes pode ser acessada simplesmente pela leitura ou escrita contínua da porta de dados.

PORTA DE COMANDO (Porta de entrada/saída &H99)

Esta porta de comando é utilizada para três propósitos:

1 - Setar o registro de endereços da porta de dados.

2 - Ler o Registro de Estado do VDP.

3 - Escrever a um dos Registros de modo do VDP.

REGISTRO DE ENDEREÇOS

O registro de endereçamento da porta de dados deve ser setado de diferentes modos, dependendo se o acesso subseqüente será de leitura ou de escrita.

Ele pode ser setado para qualquer valor entre &H0000 a &H3FFF, primeiro escrevendo-se o byte menos significativo e a seguir o byte mais significativo na porta de comando. Os bits 6 e 7 do byte mais significativo são usados pelo VDP para determinar se o registro de endereços está sendo setado para subseqüentes leituras ou escritas, conforme esquema

abaixo:

LEITURA	X X X X X X X X	Ø Ø X X X X X X
ESCRITA	X X X X X X X X	Ø 1 X X X X X X

É importante notar que nenhum outro acesso é feito para a VDP, que não seja escrevendo o byte mais significativo e o byte menos significativo, por causa da sua sincronização.

A manipulação das interrupções da ROM do MSX está continuamente lendo o registro de estado do VDP de forma que as interrupções podem ser desabilitadas, se necessário.

REGISTRO DE ESTADO DO VDP

Ler a porta de comando fará com que se conheça o conteúdo do registro de estado do VDP. Este contém diversas flags, como mostra o esquema abaixo:

7	6	5	4 3 2 1 Ø
Flag F	Flag 5S	Flag C	5 números de sprites

Os 5 bits dos números de sprites contém o número (de Ø a 31) do sprite engatilhado na flag 5S (Fifth Sprite).

A flag C, de Coincidência, normalmente vale Ø, mas passará a valer 1 se algum sprite tiver um ou mais pixéis (*Picture Element*) sobrepostos. A leitura do registro de estado irá resetar esta flag.

Note que a coincidência somente é checada quando da geração do pixel durante um quadro do vídeo, o que ocorre aproximadamente a cada 24 ms. Se os sprites se movem muito rapidamente sobre outros entre um período de checagem, não será acusada nenhuma coincidência.

A flag 5S normalmente vale Ø, mas será setada se houver mais do que quatro sprites em uma linha de pixéis. A leitura do registro de estado resetará esta flag.

A flag F (*Frame*) também normalmente vale Ø, mas é setada ao final da última linha ativa do quadro do vídeo, o que ocorre, aqui no Brasil, a cada 24 ms. A leitura do registro de estado resetará esta flag.

Existe um sinal de saída do VDP que gera interrupções no Z8Ø na mesma freqüência.

REGISTROS DE MODOS DO VDP

O VDP tem oito registros somente para escrita, ou seja, armazenamento, numerados de Ø a 7, que controlam suas operações. Um registro em particular é setado primeiro ao se escrever um byte de dados e em seguida o byte de seleção de registros para a porta de comando.

Este byte de seleção de registros contém o número do registro nos seus 3 bits mais baixos: **1ØØØØXXX**. Como os registros de modos são apenas de escrita, não podendo ser lidos, a ROM do MSX mantém uma cópia exata dos oito registros em sua área de trabalho na RAM.

Portanto, utilizando-se as rotinas padrão da ROM do MSX

para as funções do VDP teremos certeza de que esta cópia estará exatamente igual aos registros.

Registro de modo Ø

7	6	5	4	3	2	1	Ø
Ø	Ø	Ø	Ø	Ø	Ø	M3	EV

O bit EV (*External VDP*) determina se uma entrada externa ao VDP deve ser habilitada ou não:

Ø = Desabilitada
1 = Habilitada

O bit M3 (3 mode) é um dos tres bits de modo de seleção do VDP.

Registro de modo 1

7	6	5	4	3	2	1	Ø
4x16 k	BLANK	IE	M1	M2	Ø	SIZE	MAG

O bit MAG (*Magnification*) determina se os sprites serão em tamanho normal ou duplicado:

Ø = normal
1 = duplicado

O bit SIZE determina se o padrão de cada sprite será de 8x8 bits ou 16x16 bits:

Ø = 8x8
1 = 16x16

Os bits M1 e M2 determinam o modo de operação do VDP em conjunto com o bit M3 do registro de modo Ø:

M1	M2	M3	
Ø	Ø	Ø	32x24 Modo texto
Ø	Ø	1	Modo gráfico
Ø	1	Ø	Multicolorido
1	Ø	Ø	40x24 Modo texto

O bit IE (*Interrupt Enable*) habilita ou não o sinal de interrupção de saída do VDP:

Ø = Desabilita
1 = Habilita

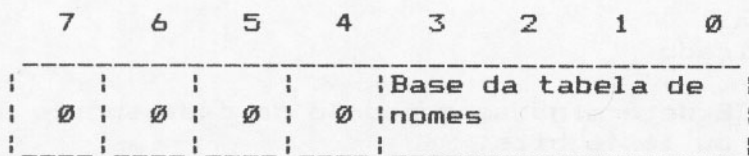
O bit BLANK é usado para habilitar ou não todo o vídeo. Quando a tela estiver vazia ela terá a mesma cor que a borda:

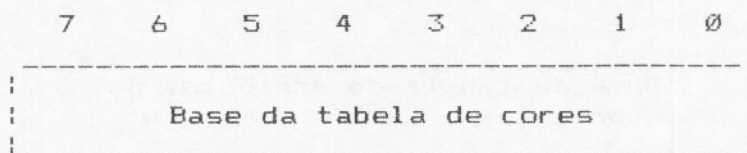
Ø = Desabilita
1 = Habilita

O bit 4/16K altera as características de endereçamento da VRAM, entre chips de 4 ou 16 K:

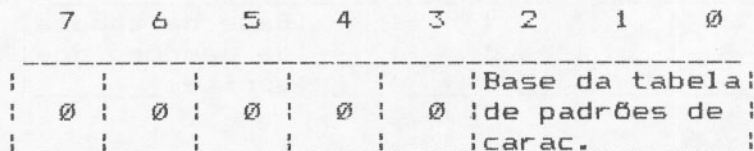
Ø = 4K
1 = 16K

Registro de modo 2

7	6	5	4	3	2	1	Ø
Ø	Ø	Ø	Ø	Base da tabela de nomes			

O registro de modo 2 define o endereço inicial da tabela de nomes na VRAM. Os 4 bits especificam apenas posições ØØBB BBØØ ØØØØ ØØØØ do endereço completo, de forma que se o registro contiver &HØF, resultará num endereço base igual a &H3CØØ.

Registro de modo 3

Este registro define o endereço inicial da tabela de cores da VRAM. Os 8 bits disponíveis especificam apenas as posições ØØBB BBBB BBØØ ØØØØ do endereço completo, de forma que se o registro contiver o valor &HFF resultará num endereço base igual a &H3FCØ. No modo gráfico apenas o bit 7 é efetivamente aquele que serve de base para &HØØØØ ou &H2ØØØ. Os bits de 6 a Ø devem ser 1.

Registro de modo 4

```
   7    6    5    4    3    2    1    Ø
 ______________________________________
|    |    |    |    |    |Base da tabela|
| Ø  | Ø  | Ø  | Ø  | Ø  |de padrões de |
|____|____|____|____|____|carac._______|
```

O registro de modo 4 define o endereço inicial da tabela de padrões de caracteres na VRAM. Os 3 bits disponíveis especificam apenas as posições ØØBB BØØØ ØØØØ ØØØØ do

endereço completo de forma que se o registro contiver &HØ7 resultará em um endereço base igual a &H38ØØ.

No modo gráfico apenas o bit 2 é aquele que oferece a base para &HØØØØ a &H2ØØØ. Os bits Ø e 1 devem ser 1.

Registro de modo 5

```
       7    6    5    4    3    2    1    Ø
     ________________________________________
     |    |Base da tabela de atributos de   |
     | Ø  | sprites.                        |
     |____|_________________________________|
```

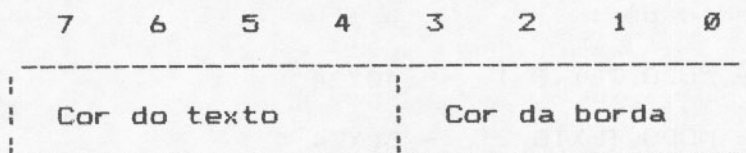
Este registro define o endereço inicial da tabela de atributos dos sprites na VRAM. Os 7 bits disponíveis especificam somente posições ØØBB BBBB BØØØ ØØØØ do endereço completo de forma que se o registro contiver o valor &HØ7 resultará num endereço base igual a &H3F8Ø.

Registro de modo 6

```
       7    6    5    4    3    2    1    Ø
     ________________________________________
     |    |    |    |    |    |Base da tabela|
     |    |    |    |    |    |de padrões dos|
     |____|____|____|____|____|sprites_______|
```

o registro de modo 6 define o endereço inicial da tabela de padrões dos sprites na VRAM. Os 3 bits disponíveis especificam apenas as posições ØØBB BØØØ ØØØØ ØØØØ do endereço completo de forma que se o registro contiver o valor &HØ7 resultará num endereço base igual a &H38ØØ.

Registro de modo 7

```
       7     6     5     4    3     2     1     Ø
     ------------------------------------------------
     |                        |                     |
     |  Cor do texto          |  Cor da borda       |
     |________________________|_____________________|
```

Os bits da cor da borda, nem é preciso dizer, determinam a cor da região em torno da área útil do vídeo, qual seja, a borda.
Até que ficou boa essa explicação!
Só faltou falar que isso vale para os quatro modos do VDP. Eles também determinam a cor de todos os pixéis que valem Ø no modo texto 4Øx24.

Os bits de cor do texto determinam a cor de todos os pixéis que valem 1 no modo texto 4Øx24. Eles não possuem nenhum efeito sobre os outros três modos, onde existe uma grande flexibilidade pela utilização da tabela de cores.

Vale repetir que as cores do VDP são:

Ø- Transparente	8- Vermelho médio
1- Preto	9- Vermelho claro
2- Verde médio	1Ø- Amarelo escuro
3- Verde claro	11- Amarelo claro
4- Azul escuro	12- Verde escuro
5- Azul claro	13- Magenta
6- Vermelho escuro	14- Cinza
7- Azul ciam	15- Branco

NOTA: Se você tiver um HOTBIT repare nas diferenças destas explicações e as contidas nas instruções de uso daquele micro, nas páginas 14Ø e 141.

MODOS DE TELA (*SCREEN MODES*)

O VDP possui 4 modos de operação, cada um com as suas particularidades:

SCREEN Ø- MODO TEXTO 1 - 4ØX24

SCREEN 1- MODO TEXTO 2 - 32X24

SCREEN 2- MODO GRÁFICO 1 - 32X24

SCREEN 3- MODO GRÁFICO 2 - MULTICOLORIDO

O elemento central do VDP, do ponto de vista do programador é a Tabela de Nomes, que é uma simples listagem de valores de 8 bits dos códigos dos caracteres armazenados na VRAM. Possui 96Ø bytes de comprimento, no modo texto 4Øx24, 768 bytes de comprimento no modo texto 32x24, modo gráfico e modo multicolorido.

Cada posição na tabela de nomes corresponde a uma locação particular na tela.

Durante um quadro do vídeo o VDP lerá seqüencialmente cada código de caractere da Tabela de Nomes, começando pelo endereço inicial. Assim que cada código de caractere correspondente a um padrão de 8x8 pixéis é lido, a tabela de padrões de caracteres é consultada, para se verificar qual o padrão do caractere em questão, para que finalmente possa ser enviado para o vídeo.

O aparecimento na tela pode ser alterado tanto pelas mudanças nos códigos de caracteres da tabela de nomes quanto pelas mudanças dos padrões de pixéis da tabela de padrões de caracteres.

SCREEN Ø - MODO TEXTO 1 - 4ØX24

A tabela de nomes ocupa 96Ø bytes da VRAM, de &HØØØØ a

&HØ3BF:

```
               _______________________________
     &HØØØØ|||||||||||              __________| Ø
     &HØØ28|||||||                    ______| 1
     &HØØ5Ø|||||||                       ____| 2
     &HØØ78||||||                          __| 3
       ...   ||                               |..
       ...                                    ...
       ...                                    |..
     &HØ348|                              _____|21
     &HØ37Ø|                          _________|22
     &HØ398|_______________________________|23
           Ø123456789Ø1........89Ø123456789Ø
```

A tabela de padrões de caracteres ocupa 2Kb da VRAM, de &HØ8ØØ a &HØFFF. Cada bloco de 8 bytes contém o padrão dos pixéis para um código de caractere:

```
Ø  Ø  Ø  Ø  Ø  Ø  Ø  Ø  byte Ø
Ø  Ø  1  1  1  1  Ø  Ø  byte 1
Ø  Ø  1  1  1  1  Ø  Ø  byte 2
Ø  Ø  Ø  1  1  Ø  Ø  Ø  byte3
Ø  Ø  Ø  1  1  Ø  Ø  Ø  byte 4
Ø  Ø  Ø  1  1  Ø  Ø  Ø  byte 5
Ø  Ø  Ø  1  1  Ø  Ø  Ø  byte 6
Ø  Ø  Ø  1  1  Ø  Ø  Ø  byte 7
```

O primeiro bloco contém o padrão para o código de caractere Ø, o segundo, para o código de caractere 1, e assim por diante, até o código 255.

Vale lembrar que apenas os 6 bits mais significativos são mostrados na tela neste modo. As cores dos pixéis Ø e 1 neste modo são definidas pelo registro de modo 7.

Inicialmente elas são azul e branca

SCREEN 1 - MODO TEXTO 2 - 32x24

A tabela de nomes ocupa 768 bytes da VRAM, de &H18ØØ até &H1AFF. Da mesma forma que no modo texto normal (4Øx24), aqui também os respectivos códigos dos caracteres são colocados nas devidas posições da tabela. A declaração VPOKE pode ser utilizada para se obter familiaridade com o layout da tela:

```
         ________________________________
&H18ØØ  |||||||__________________________|  Ø
&H182Ø  |||||    ________________________|  1
&H184Ø  ||             __________________|  2
&H186Ø  ||                   ____________|  3
  ...   |                                |  .
  ...   |                                   .
  ...                                       .
&H1AAØ        ______________|||||||||||||  21
&H1ACØ  |   _______________________||||||||||  22
&H1AEØ  |________________________________|  23
         Ø123456789Ø123456789Ø123456789Ø1
```

A tabela de padrões dos caracteres ocupa 2Kb da VRAM, de HØØØØ até HØ7FF. A sua estrutura é a mesma que o modo texto anterior, com a diferença de que todos os pixéis de um padrão de 8x8 são impressos na tela.

A cor da borda é definida pelo registro de modo 7 do VDP. Uma tabela adicional, a tabela de cores, determina a cor dos pixéis Ø e 1. Esta ocupa 32 bytes da VRAM, de H2ØØØ até H2Ø1F.

Cada entrada nessa tabela de cores define as cores dos pixéis Ø e 1 para um grupo de 8 códigos de caracteres, sendo que os 4 bits menos significativos definem a cor do pixel Ø, e os outros 4 bits definem a cor do pixel 1.

A primeira entrada na tabela define a cor dos códigos de

caracteres de Ø a 7, a segunda, dos códigos de 8 a 15, e assim por diante, até 32 entradas.

SCREEN 2 - MODO GRAFICO 1

A tabela de nomes ocupa 768 bytes da VRAM, de &H18ØØ até &H1AFF. A tabela é inicializada com a seqüência de códigos de caracteres de Ø a 255, repetida três vezes, sem ser alterada. Neste modo, a tabela de padrões de caracteres é que é modificada durante as operações normais.

A tabela de padrões de caracteres ocupa 6Kb da VRAM, de &HØØØØ até &H17FF. Embora a sua estrutura seja a mesma que no modo texto, ela não contém o conjunto de caracteres, mas é inicializada com Ø em todos os pixéis. Os primeiros 2Kb desta tabela são endereçados pelos códigos de caracteres do primeiro terço da tabela de nomes; os segundos 2Kb, pelo terço central da tabela de nomes, e os últimos 2Kb, pelo último terço da tabela de nomes.

Por causa dos padrões seqüenciais da tabela de nomes, toda a tabela de padrões dos caracteres é lida linearmente, durante um quadro do vídeo.

```
        ________________________________
 &HØØØØ|                                |  Ø
 &H1ØØØ|                                |  1
 &H2ØØØ|                                |  2
 &H3ØØØ|                                |  3
 &H4ØØØ|                                |  4
   ... |                                |
   ... |                                |
   ... |                                |
 &H15ØØ|                                | 21
 &H16ØØ|                                | 22
 &H17ØØ|________________________________| 23
        Ø1234567890123456789Ø12345678901
```

A cor da borda é definida pelo registro de modo 7 do VDP, inicialmente azul. A tabela de cores ocupa 6Kb da VRAM, de &H2000 até &H37FF.

Existe um mapeamento idêntico, byte a byte, entre a tabela de padrões de caracteres e a tabela de cores, mas, pelo fato de ser necessário 1 byte para definir as cores dos pixéis 0 e 1, a resolução de cores dos pixeís é baixa. Os 4 bits menos significativos de uma entrada na tabela de cores definem a cor de todos os pixéis 0 da linha de 8 pixéis correspondentes. Os 4 bits mais significativos definem a cor dos pixéis 1.

A tabela de cores é inicializada de tal forma que tanto a cor dos pixéis 0, como a cor dos pixéis 1 é azul, para ela inteira. Por esse fato, faz-se necessário alterar uma das cores quando um bit é setado na tabela de padrões de caracteres.

SCREEN 3 - MODO GRAFICO 2

A tabela de nomes ocupa 768 bytes da VRAM, de &H0800 até &H0AFF, e o mapeamento da tela é o mesmo que o do modo texto 2, screen 1.

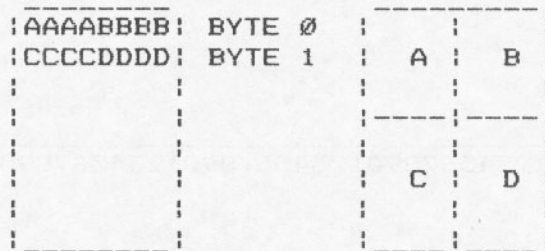
A tabela é inicializada com os seguintes padrões de códigos de caracteres:

&H00 a &H1F	repetido 4 vezes
&H20 a &H3F	repetido 4 vezes
&H40 a &H5F	repetido 4 vezes
&H60 a &H7F	repetido 4 vezes
&H80 a &H9F	repetido 4 vezes
&HA0 a &HBF	repetido 4 vezes

Da mesma forma que no modo gráfico anterior, é a tabela de padrões de caracteres que é modificada durante uma operação normal.

A tabela de padrões de caracteres ocupa 1536 bytes da VRAM, de &HØØØØ a &HØ5FF.

Como nos outros modos, há uma associação entre o código de um caractere e um bloco de 8 bytes da tabela de padrões dos caracteres. Por causa da baixa resolução deste modo, apenas 2 bytes do bloco de padrões são necessários para definir um padrão de um bloco de 8x8:

```
 ________          __________
|AAAABBBB| BYTE Ø |    |    |
|CCCCDDDD| BYTE 1 |  A | B  |
|        |        |    |    |
|        |        |----|----|
|        |        |    |    |
|        |        |  C | D  |
|        |        |    |    |
|________|        |____|____|
```

Como pode ser visto no esquema acima, cada 4 bits do bloco de 2 bytes define a cor do código, e isto define a COLOUR do bloco de 8x8 pixéis.

Dessa forma, a entrada de 8 bytes de um bloco de padrões, que pode ser utilizada por um determinado código de caractere, utilizará seções de 2 bytes diferentes, dependendo da locação do caractere na tela, ou a sua posição na tabela de nomes:

```
Linhas da tela Ø,4,8,12,16,2Ø    bytes Ø e 1
               1,5,9,13,17,21    bytes 2 e 3
               2,6,1Ø,14,18,22   bytes 4 e 5
               3,7,11,15,19,23   bytes 6 e 7
```

Quando a tabela de nomes for preenchida com essa seqüência de códigos de caracteres mostrada acima, a tabela de padrões de caracteres será lida linearmente durante um quadro do vídeo:

```
         ________________________________
&H0000|                                  | 0
&H0002|                                  | 1
&H0004|                                  | 2
&H0006|                                  | 3
&H0008|                                  | 4
   ...|                                  |
   ...|                                  |
   ...|                                  |
&H0502|                                  | 21
&H0504|                                  | 22
&H0506|__________________________________| 23
       01234567890123456789012345678901
```

A cor da borda é definida pelo registro de modo 7 do VDP, e é inicialmente azul.

Não há tabela de cores separada, visto que as cores são definidas diretamente pelo conteúdo da tabela de padrões de caracteres que são inicialmente preenchidos com azul.

SPRITES

O VDP pode controlar 32 sprites em qualquer modo, exceto no modo texto 40x24, screen 0.

O seu tratamento é idêntico em todos os modos e independente de qualquer orientação dada para caracteres.

A tabela de atributos de sprites ocupa 128 bytes da VRAM, de &H1B00 a &H1BFF. A tabela contém 32 blocos de 4 bytes, um para cada sprite. O primeiro bloco controla o sprite 0 (sprite de topo), o segundo bloco controla o sprite 1, e assim até o sprite 31. O formato de cada

bloco é mostrado a seguir:

Posição vertical		B0
Posição Horizontal		B1
Número do padrão (nome)		B2
EC 0 0 0	Código da cor	B3

O byte 0 especifica a coordenada do pixel do canto superior esquerdo do sprite (Y).

O sistema de coordenadas trabalha de -1 (&HFF), para a linha de topo da tela, até 190 (&HBE), para a linha inferior. Valores menores que -1 podem ser utilizados para projetar o sprite, a partir do topo da tela.

Note que não existe associação entre este sistema de coordenadas e o sistema de coordenadas gráficas, e por isso um sprite será sempre 1 pixel mais baixo que seu equivalente ponto gráfico. O valor 208 (&HD0), num bloco de atributos de sprites fará com que o VDP ignore aquele sprite, fazendo com que ele não mais apareça na tela.

O byte 1 especifica a coordenada X horizontal do pixel situado no canto superior esquerdo do sprite. A coordenada X varia de &H00, para o pixel mais a esquerda, até &HFF (255), para o pixel mais a direita.

O byte 2 seleciona um dos 256 padrões de 8x8 bits disponíveis na tabela de padrões de caracteres. Se o bit de tamanho (*SIZE*) do registro de modo 1 do VDP estiver setado, resultará em um padrão de 16x16 bits, ocupando 32 bytes cada; os 2 bits menos significativos do número

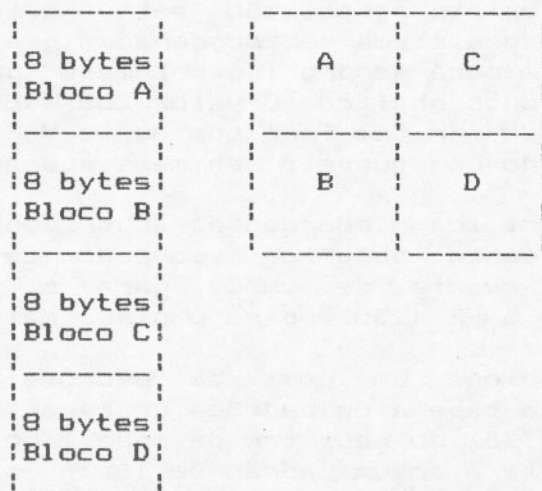
do padrão são ignorados. Portanto os números de padrão Ø,1,2 e 3 selecionam o padrão de número Ø.

No byte 3, os 4 bits do código da cor definem a cor dos pixéis 1 do padrão do sprite, e os pixéis Ø são sempre transparentes.

O bit EC (*Early Clock*) normalmente vale Ø, mas se estiver setado, fará com que a coluna do sprite seja equivalente à posição (X-32), sendo dessa forma possível inserir um sprite antes da primeira coluna.

A tabela de padrões dos sprites ocupa 2Kb da VRAM, de &H38ØØ até &H3FFF. Ela contém 256 padrões de 8x8 pixéis, numerados de Ø a 255.

Se o bit SIZE do registro de modo 1 do VDP for Ø, resultará em sprites 8x8, fazendo com que cada bloco de 8 bytes do padrão de um sprite seja estruturado da mesma forma que a tabela de padrões de caracteres do modo SCREEN Ø. Se o bit SIZE for 1, resultará em sprites de 16x16, sendo então necessários 4 bytes para definir seu padrão, conforme esquema abaixo:

```
 --------     ----------------
|        |   |       |        |
|8 bytes |   |   A   |   C    |
|Bloco A |   |       |        |
|________|   |_______|________|
|        |   |       |        |
|8 bytes |   |   B   |   D    |
|Bloco B |   |       |        |
|________|   |_______|________|
|        |
|8 bytes |
|Bloco C |
|________|
|        |
|8 bytes |
|Bloco D |
|________|
```

CAPÍTULO 28 — O PSG (*PROGRAMMABLE SOUND GENERATOR* OU GERADOR DE SONS PROGRAMÁVEL)

Além de controlar 3 canais de som, o chip 8910 contém duas portas de dados de 8 bits, chamadas de porta A e B, que servem para o interfaceamento do joystick e a entrada do cassete.

O PSG aparece para o Z80 como sendo três portas de entrada/saída, chamadas Porta de Endereços, Porta de Escrita de dados e Porta de Leitura de dados.

Porta de endereços (Porta de entrada/saída &HA0):

O PSG contém 16 registros internos que definem completamente seu modo de operação.

Um registro específico é selecionado, pela escrita do seu número, de 0 a 15 para esta porta. Uma vez

selecionado, o acesso a este registro pode ser repetido inúmeras vezes, via duas portas de dados.

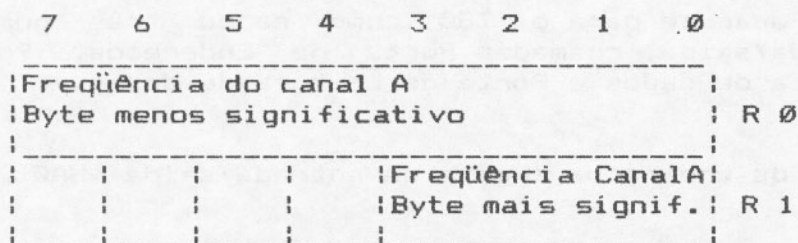
Porta de Escrita de dados (Porta de entrada/saída &HA1).

Esta porta é utilizada para escrever dados para qualquer registro que tenha sido selecionado pela porta de endereços.

Porta de Leitura de dados (Porta de entrada/saída &HA2).

Esta porta é utilizada para ler dados de qualquer registro que tenha sido selecionado pela porta de endereços.

Registros Ø e 1

```
   7    6    5    4    3    2    1    Ø
 ___________________________________
|Freqüência do canal A              |
|Byte menos significativo           | R Ø
|___________________________________|
|    |    |    |    |Freqüência CanalA|
|    |    |    |    |Byte mais signif.| R 1
|____|____|____|____|_________________|
```

Estes dois registros são usados para definir a freqüência do gerador de sons do canal A.

São produzidas inúmeras freqüências, pela divisão de uma freqüência mestra fixa pelo número armazenado nos registros Ø e 1, que pode estar na faixa de Ø a 4Ø95.

O PSG divide uma freqüência externa de 1,7897725 Mhz por 16, para produzir a freqüência mestra de geração de tons, que vale 111,861 Hz. A freqüência de saída do gerador de sons pode, portanto, estar na faixa de 111,861 Hz (dividida por 1), até 27,3 Hz (dividida por 4Ø95).

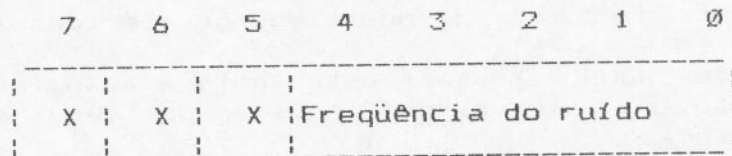
Registros 2 e 3

Estes dois registros controlam o gerador de sons do canal B de forma idêntica à do canal A, descrito acima.

Registros 4 e 5

Estes dois registros controlam o gerador de sons do canal C da mesma forma que o controle do canal A descrito anteriormente.

Registro 6

7	6	5	4	3	2	1	Ø
X	X	X	Freqüência do ruído				

Em adição aos três geradores de sons de ondas quadradas, o PSG possui um gerador de ruídos simples.

A freqüência dessa fonte de ruídos pode ser controlada de um modo similar ao dos geradores de sons.

Os 5 bits menos significativos do registro 6 armazenam um divisor, de Ø a 31, para a mesma freqüência mestra de

111,861 Hz.

Registro 7

7	6	5	4	3	2	1	Ø
dir port B	dir port A	Ruíd C	Ruíd B	Ruíd A	Tom C	Tom B	Tom A

Este registro habilita ou não os geradores de sons ou de ruídos, para cada um dos três canais:

Ø = Habilita
1 = Desabilita

Também controla a direção da interface das portas A e B, por onde os joysticks e o cassete são conectados:

Ø = Entrada
1 = Saída

O registro 7 sempre deve conter 1ØXXXXXX, ou poderá ser danificado, já que sempre estão conectados aos seus pinos de entrada/saída, dispositivos periféricos ativos.

A declaração "SOUND" forçará estes bits a assumirem seu valor padrão, mas a nível de linguagem de máquina, não existe proteção.

Registro 8

7	6	5	4	3	2	1	Ø
X	X	X	Modo	Amplitude do canal A			

Os 4 bits de amplitude determinam a amplitude do canal A, de um mínimo de Ø até um máximo de 15.

O bit MODO seleciona amplitude modulada (1) ou fixa (Ø). Quando é selecionada a amplitude modulada o valor da amplitude fixa é ignorado e o canal é modulado pela saída do gerador de envelopes.

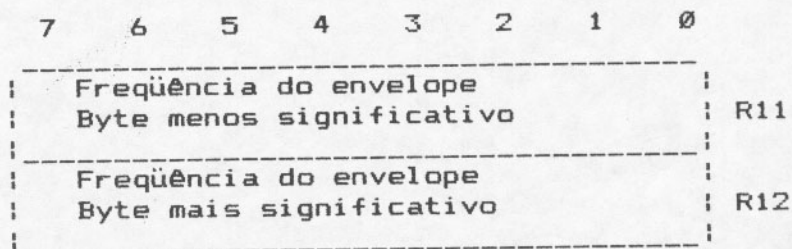
Registro 9

Este registro controla a amplitude do canal B, da mesma forma que o canal A.

Registro 1Ø

Este registro controla a amplitude do canal C, da mesma forma que o canal A.

Registros 11 e 12

```
   7    6    5    4    3    2    1    Ø
 ______________________________________
 |  Freqüência do envelope             |
 |  Byte menos significativo           |  R11
 |_____________________________________|
 |  Freqüência do envelope             |
 |  Byte mais significativo            |  R12
 |_____________________________________|
```

Estes dois registros controlam a freqüência do gerador de envelopes simples, utilizados para modulações de amplitudes.

Da mesma forma que nos geradores de sons, a freqüência é

determinada pela colocação de um divisor nos registros, cujo valor está na faixa de 1 a 65535, com o registro 11 armazenando os 8 bits menos significativos e o registro 12 armazenando os 8 bits mais significativos.

A freqüência mestra para o gerador de envelopes vale 6991 Hz, fazendo com que a faixa de freqüências de envelopes varie de 6991 Hz (dividida por 1) até 0,11 Hz (dividida por 65535).

Registro 13

```
    7    6    5    4    3   2    1    0
 ______________________________________
 |    |    |    |    |Forma do envelope |
 | X  | X  | X  | X  |                  |
 |____|____|____|____|__________________|
```

Os 4 bits de Forma de Envelopes determinam a forma da modulação da amplitude do envelope produzido pelo gerador de envelopes:

```
  3     2     1     Ø     Modulação do Envelope
 ______________________
|                      |  |\
|  Ø     Ø     X     X |  | \______________
|                      |
|                      |   /|
|  Ø     1     X     X |  / |______________
|                      |
|                      |  |\ |\ |\ |\ |\ |
|  1     Ø     Ø     Ø |  | \| \| \| \| \|
|                      |
|                      |  |\
|  1     Ø     Ø     1 |  | \______________
|                      |
|                      |
|  1     Ø     1     Ø |   /\   /\   /\   /\
|                      |  /  \/  \/  \/  \
|                      |     ______________
|  1     Ø     1     1 |  \ |
|                      |   \|
|                      |
|  1     1     Ø     Ø |   /| /| /| /| /| /
|                      |  / |/ |/ |/ |/ |/
|                      |    ______________
|  1     1     Ø     1 |   /
|                      |  /
|                      |
|  1     1     1     Ø |  \  /\   /\   /\  /
|                      |   \/  \/   \/  \/
|                      |
|  1     1     1     1 |   /|
|______________________|  / |______________
```

Registro 14

7	6	5	4	3	2	1	Ø
K7 Entr	Modo do Tec.	Joy Trg. B	Joy Trg. A	Joy Dir.	Joy Esq.	Joy Trás	Joy Frente

Este registro é usado para ler a entrada da porta A do PSG. Os 6 bits de joystick refletem o estado dos botões das 4 direções dos 2 botões trigger:

Ø = pressionado
1 = solto

Alternativamente até 6 paddles podem ser conectados, em vez de joysticks.

Embora a maioria das máquinas MSX possuam 2 entradas de 9 pinos para joystick, apenas uma pode ser lida de cada vez, a que é selecionada pelo bit seletor de joystick do registro 15.

O bit de modo de teclado não é utilizado na maioria das máquinas.

O bit da entrada do cassete é utilizado para ler o sinal vindo da saída EAR do cassete, que passa por um comparador que o converte para sinais digitais.

Registro 15

7	6	5	4	3	2	1	Ø
LED Kana	Sel. Joy.	Pulso 2	Pulso 1	1	1	1	1

Este registro é utilizado como saída da porta B do PSG.

Os 4 bits menos significativos são conectados via buffers abertos TTL aos pinos 6 e 7 de cada conector de joystick. Normalmente valem 1, quando um joystick ou paddle estão conectados, e possuem portanto função de saída.

Os 2 bits de pulso são utilizados para gerar um pulso positivo curto para qualquer paddle conectado a qualquer porta de joystick. Cada paddle possui um circuito próprio que controla o comprimento do pulso.

O bit seletor de joystick determina qual conector de joystick está conectado à porta A do PSG para saída:

Ø = Conector 1
1 = Conector 2

O bit LED Kana é utilizado por algumas máquinas para indicar em que estado se encontra o teclado naquele instante.

CAPÍTULO 29 — ROM BIOS ASSOCIADOS AO USO DE SLOTS

A partir deste capítulo, até o de número 34, faremos um estudo detalhado da ROM BIOS, que significa, como já vimos anteriormente, **BASIC INPUT/OUTPUT SYSTEM.**

Neste ponto, você necessariamente deve ter entendido todo o grupo de instruções do poderoso Z8Ø, para que possa aplicar nas suas rotinas ou nos seus programas.

A grande maioria dos livros que rodam por aí, quando chega em capítulos semelhantes a estes, diz que não explicará nada a respeito das instruções do Z8Ø.

Esta é uma grande vantagem deste livro: você não precisa adquirir nenhum outro para se aprofundar em seus estudos. Tudo o que você precisa saber sobre seu micro está aqui.

Com o propósito de facilitar seus estudos, bem como facilitar o meu trabalho de escrever esta enciclopédia, nestes capítulos eu padronizei o formato da apresentação

das rotinas do BIOS, da seguinte maneira:

A primeira linha contém o nome da rotina. Lembre-se de que este nome não tem nada a ver com as palavras - chaves em Basic, mas são apenas referências à seu processamento, pois seus nomes são abreviaturas do que fazem, e são baseadas na terminologia Microsoft; a linha seguinte mostra a instrução do Z80 que deve ser executada para acessar aquela rotina.

Em seguida, vem a descrição do que a rotina executa, e as informações necessárias para passar ou receber parâmetros dela.

A parte seguinte contém uma listagem dos registros do Z80 e locações da memória que podem ser alterados por aquela rotina, para terminar, às vezes, com alguma informação adicional.

Estes capítulos estão diretamente associados aos Apêndices E e F, que tratam dos "HOOKS" (ganchos), e às variáveis do sistema.

Os pontos de entrada do BIOS descritos neste capítulo são utilizados para gerenciar o sistema de slots do seu micro.

RDSLT (*Read Slot*) &H000C
CALL &H0024

Esta chamada é utilizada para ler uma posição da memória em qualquer slot.

Parâmetros de entrada

O par de registros HL deve conter o endereço da locação a ser lida, e o registro A deve especificar de qual slot

vai ser lido. O valor do acumulador tem o seguinte significado: os 2 bits menos significativos (Ø e 1) contêm o número do slot primário (Ø a 3) a ser usado, os próximos 2 bits contém o número do slot secundário (Ø a 3), os próximos 3 bits não são utilizados e se um slot secundário foi especificado, então o bit 7 deve ser setado e de outra forma, resetado.

Parâmetros de saída

O registro A armazena o conteúdo da locação da memória especificada.

Alterações

Registros AF, BC e DE são afetados.

NOTA : Esta rotina desabilita todas as interrupções. Atenção ao selecionar a página 3 de um slot usando esta rotina: você causará um "crash" no sistema! Veja em ENASLT a solução desse problema.

WRSLT (*Write slot*) &HØØ14
CALL &HØØ14

Esta chamada é usada para escrever em qualquer posição da memória de qualquer slot.

Parâmetros de entrada

O par de registros HL deve conter o endereço da locação onde será escrito um dado, o acumulador deve conter o número especificando o slot, e o registro E, o valor a ser escrito.

Parâmetros de saída

Não há.

Alterações

Registros AF, BC e D são afetados por esta chamada.

NOTA : Idêntica à anterior.

CALSLT (*Call Slot*) &HØØ1C
CALL &HØØ1C

Esta rotina executa uma chamada inter slots selecionando uma página de qualquer um deles e então chamando um endereço daquela página.

Parâmetros de entrada

O par de registros IX deve conter o endereço que deverá ser chamado, e os bits mais significativos do par IY devem conter a especificação de qual slot. O formato requerido para o byte mais significativo do par de registros IY deve ser idêntico ao do acumulador na rotina RDSLT.

Parâmetros de saída

Não existem, a menos que sejam criados pela chamada. Estes podem retornar em qualquer registro, exceto o A', que armazena o estado do mecanismo de seleção de slot após a chamada.

Alterações

Registros AF', BC', DE' e HL' são alterados.

NOTA : Idêntica à anterior. Esta chamada também pode ser feita via instruções RST do Z 8Ø.

ENASLT (*Enable Slot*) &HØØ24

CALL &HØØ24

Esta chamada seleciona uma página de um slot.

Parâmetros de entrada

Os 2 bits mais significativos do registro H são usados para selecionar a página apropriada, da seguinte maneira:

BITS	PÁGINA SELECIONADA
ØØ	&HØØØØ-&H3FFF
Ø1	&H4ØØØ-&H7FFF
1Ø	&H8ØØØ-&HBFFF
11	&HCØØØ-&HFFFF

O acumulador deve especificar o slot a ser selecionado.

Parâmetros de saída

Não há.

Alterações

Registros AF, BC, DE e HL são afetados por esta chamada.

NOTA : Esta rotina desabilita todas as interrupções. O problema da não habilitação para seleção da página 3 de um slot, nas rotinas RDSLT, WRSLT e CALSLT pode ser resolvido utilizando-se esta rotina. Para ler o endereço &HDØØØ no slot 3, utilize os códigos a seguir, como exemplo:

```
CALL &HØ138       Lê o registro de seleção de
                  slot primário
PUSH AF           Salva conteúdo de AF
LD HL, &HDØØØ     Endereço a ser lido
PUSH HL           Salva este endereço
LD A, 3
DI                As interrupções devem ser
                  desabilitadas pois alteram
```

```
                    a página 3
CALL &H0024         Habilita página 3 do slot 3
POP HL              Recupera endereço
LD H, (HL)          Armazena conteúdo desejado
POP AF
CALL &H013B         Re-habilita página 3 do
                    sistema
EI
LD A, H             Retorna valor desejado no
                    acumulador
RET
```

Métodos similares podem ser utilizados para escrever e chamar sub-rotinas na página 3 de outros slots.

RSLREG (*Read Slot Register*) &H0138
CALL &H0138

Esta chamada lê o conteúdo do registro de seleção de slot primário.

Parâmetros de entrada

Não existem.

Parâmetros de saída

Uma cópia do conteúdo do registro de seleção de slot primário é armazenada no acumulador.

Alterações

Apenas o registro A é alterado.

NOTA : Esta rotina é simplesmente uma instrução IN (provavelmente IN A, &HA8), seguida de uma instrução RET.

WSLREG (*Write Slot Register*) &HØ13B
CALL &HØ13B

Esta chamada envia dados ao registro de seleção de slot primário.

Parâmetros de entrada

O registro A deve conter o valor a ser enviado.

Parâmetros de saída

Não há.

Alterações

Não há.

NOTA : Esta rotina é simplesmente uma instrução OUT (provavelmente OUT &HA8), seguida da instrução RET.

CALBAS (*Call Basic*) &HØ159
CALL &HØ159

Este ponto de entrada é utilizado pelo interpretador Basic para executar uma chamada inter slot num cartucho de expansão do interpretador Basic.

Parâmetros de entrada

O endereço a ser chamado é armazenado no registro IX.

Parâmetros de saída

Dependem da chamada inter slot.

CAPÍTULO 3Ø — ROM BIOS ASSOCIADOS AO CONSOLE

Este capítulo descreve aquelas chamadas que são utilizadas para controlar o console, isto é, o teclado, a impressora e o display de texto.

CHSNS &H009C

CALL &HØØ9C

Esta chamada executa duas operações. Primeiramente checa o estado das teclas SHIFT. Se uma delas estiver pressionada, e as teclas de função estiverem ativas, estas, de 6 a 1Ø, serão mostradas na tela.

Em seguida, esta rotina checa o estado do buffer do teclado.

Parâmetros de entrada

Não há.

Parâmetros de saída

Se o buffer de teclado contiver dados, a flag Z será setada.

Alterações

Somente o acumulador e as flags são alterados pela rotina.

NOTA : Esta chamada habilita as interrupções.

CHGET (*Character Get*) &H009F
CALL &H009F

Esta rotina retorna um caractere do buffer do teclado. Se o buffer contiver dados a rotina aguarda pela pressão de uma tecla, mostrando se o cursor está disponível.

Parâmetros de entrada

Não há.

Parâmetros de saída

O acumulador armazena o código do caractere cuja tecla está sendo pressionada.

Alterações

O acumulador e as flags são alterados pela chamada.

NOTA : Esta rotina chama o HOOK H-CHGE imediatamente após salvar na pilha os registros HL, DE e BC, nesta ordem. Isto permite que outros dispositivos de entrada do console sejam utilizados.

CHPUT (*Character Output*) &HØØA2
CALL &HØØA2

Esta chamada envia um caractere ao console, movendo para a próxima linha e rolando (*scroll*) a tela quando necessário. Os códigos de controle 7-13 e 27-31 são admitidos.

Parâmetros de entrada

O registro A deve conter o código do caractere a ser enviado. A posição do cursor na tela é armazenada nas variáveis CSRX, CSRY.

Parâmetros de saída

Na saída, a posição do cursor (CSRX, CSRY) é atualizada.

Alterações

Nenhum registro é alterado, mas as locações da memória CSRX, CSRY e TTYPOS são.

NOTA: Após salvar os conteúdos dos registros HL, DE, BC e AF na pilha, esta rotina chama o HOOK H-CHPU, habilitando para uso outros dispositivos do console, como interface de 8Ø colunas. Esta rotina não faz nada em modo gráfico. As interrupções são habilitadas por esta chamada.

LPTOUT (*Lprint Out*) &HØØA5
CALL &HØØA5

Esta rotina envia um caractere para a impressora, se esta estiver conectada. Se a impressora não estiver pronta para receber um código, a rotina irá aguardar até a impressora estar pronta, a menos que se pressionem as teclas CONTROL + STOP.

Parâmetros de entrada

O registro A deve conter o código do caractere a ser impresso.

Parâmetros de saída

A flag de transporte será resetada se o caractere for impresso, e setada se a tecla STOP for usada para abortar a operação.

Alterações

As flags são alteradas por esta chamada.

NOTA: A primeira ação desta rotina é chamar o HOOK H-LPTO, habilitando o envio dos dados no processamento. Um exemplo disto é programar o HOOK de tal forma que ignore os caracteres enviados para a impressora, via

```
H-LPTO  INC SP
        INC SP
        RET
```

O HOOK H-LPTO localiza-se em &HFFB6, de forma que o exemplo acima pode ser aplicado utilizando-se declarações Basic:

```
        POKE &HFFB6, &H33
        POKE &HFBB7, &H33
```

(Para ter a impressora trabalhando novamente digite POKE &HFBB6, &HC9).

LPTSTT (*Line Printer Status*) &HØØA8
CALL &HØØA8

Esta chamada checa se a impressora está pronta para receber um caractere.

Parâmetros de entrada

Não há.

Parâmetros de saída

Se a impressora estiver pronta, o registro A armazenará 255 e a flag Z será resetada.

Alterações

Somente o acumulador e as flags são afetadas por esta rotina.

CNVCHR (*Convert Character*) &HØØAB
CALL &HØØAB

Esta chamada converte um código de caractere num número de padrão que o represente perante o VDP. Para códigos de caracteres de 32 a 255, estes valores são iguais, mas os códigos de Ø a 31, de controle, cujos padrões representam os caracteres obtidos pela pressão da tecla GRAPH, são representados por 2 códigos de caracteres - código de controle 1 seguido por um caractere na faixa 64 a 95.

Parâmetros de entrada

O acumulador deve armazenar o caractere a ser convertido.

Parâmetros de saída

4 diferentes resultados podem ser obtidos por esta chamada, dependendo do estado da variável de sistema GRPHED (*Graphic Header*) e o acumulador.
1- Se GRPHED contiver Ø e o acumulador contiver o código de caractere 1, este permanecerá inalterado, GRPHED será setado e as flags de transporte e zero serão resetadas.
2- Se GRPHED contiver Ø e o acumulador armazenar um valor diferente de 1, então GRPHED e o registro A não serão alterados e as Flags de transporte e zero serão

setadas.
3- Se GRPHED contiver um número diferente de zero, e o acumulador contiver um número na faixa de 64 a 95, GRPHED será zerada, o acumulador não será alterado e as flags de transporte e zero serão setadas.
4- Se GRPHED contiver um número diferente de zero e o registro A contiver um número fora da faixa de 64 a 95, GRPHED será zerada, o acumulador não será alterado e as flags de transporte e zero serão setadas.

Alterações

Podem ser alterados por esta chamada o acumulador, as flags e a locação da memória GRPHED.

PINLIN &HØØAE
CALL &HØØAE

Esta rotina é similar à próxima, INLIN, mas, quando o micro estiver no modo AUTO, para numeração de linhas, pressionando-se (CTRL-U) não deletará o número da linha.

Parâmetros de entrada

Não há.

Parâmetros de saída

O par de registros HL deve apontar para a locação da memória abaixo do primeiro caractere no buffer, e a flag de transporte será setada se (CTRL+STOP) forem pressionadas.

Alterações

Os registros AF, BC, DE e HL podem ser modificados por esta chamada.

INLIN (*Input line*) &HØØB1
CALL &HØØB1

Esta rotina é utilizada pelo interpretador Basic para aceitar uma linha a partir do teclado, mostrando os caracteres da forma como foram digitados, e armazenando-a num buffer, até que RETURN ou CTRL+STOP sejam pressionadas.

Parâmetros de entrada

Não há.

Parâmetros de saída

O par de registros HL deve apontar para uma locação da memória abaixo do primeiro caractere do buffer e a flag de transporte será setada se CTRL+STOP forem pressionadas.

Alterações

Os registros AF, BC, DE e HL podem ser alterados.

QUINLIN (*Question mark in line*) &HØØB4
CALL &HØØB4

Esta rotina é similar à INLIN, mas é usada pela declaração INPUT para imprimir um ponto de interrogação e então aceitar uma linha de entrada a partir do teclado.

Parâmetros de entrada

Não há.

Parâmetros de saída

O par de registros HL deve apontar para a locação da memória abaixo do primeiro caractere do buffer e a flag de transporte será setada se CTRL+STOP forem pressionadas.

Alterações

Os registros AF, BC, DE e HL podem ser alterados por esta rotina.

BREAKX &HØØB7
CALL &HØØB7

Esta chamada checa para saber se CTRL+STOP estão sendo pressionadas.

Parâmetros de entrada

Não há.

Parâmetros de saída

Se as teclas estão sendo pressionadas, a flag de transporte será setada.

Alterações

O registro A e as flags são afetados por esta rotina.

ISCNTC &HØØBA
CALL &HØØBA

Esta chamada checa se a tecla STOP ou as teclas CTRL+STOP foram pressionadas. Se a tecla STOP está sendo pressionada, o programa é interrompido momentaneamente, até que seja pressionada novamente. Se as teclas CTRL+STOP forem pressionadas, o micro executará sua

sequência de BREAK, isto é, mudará para modo texto, habilitando o slot contendo o interpretador Basic e entrando em modo de comando.

Parâmetros de entrada

Não há.

Parâmetros de saída

Não há.

Alterações

O registro A, as flags e as locações da memória INTFLG e KILBUF serão afetados.

NOTA: As interrupções podem ser habilitadas se esta rotina estiver sendo utilizada.

CKCNTC &HØØBD
CALL &HØØBD

Esta rotina executa tarefa idêntica à anterior, só que um pouco mais lenta. Portanto, se for o caso, utilize a anterior.

Parâmetros de entrada

Não há.

Parâmetros de saída

Não há.

Alterações

Idênticas às anteriores.

BEEP &HØCØØ
CALL &HØCØØ

Esta rotina causará um BEEP se CTRL e a tecla G forem pressionadas.

Parâmetros de entrada

Não há.

Parâmetros de saída

Não há.

Alterações

Os registros AF, BC DE e HL e as locações da memória MUSICF, PLYCNT, VCBA a VCBA+4, VCBB a VCBB+4 e VCBC a VCBC+4 podem ser alterados por esta chamada.

CLS (*Clear the screen*) &HØØC3
CALL &HØØC3

Esta chamada limpa a tela, incluindo modos gráficos.

Parâmetros de entrada

A tela será limpa somente se a flag Zero estiver setada.

Parâmetros de saída

Não há.

Alterações

Os registros AF, BC e DE, e as locações da memória LINTTB, CSRX e CSRY podem ser alterados por esta rotina.

POSIT (*Position*) &HØØC6
CALL &HØØC6

Esta rotina move o cursor para uma posição especificada.

Parâmetros de entrada

O registro H deve armazenar o número da coluna desejada e o registro L deve armazenar o número da linha.

Parâmetros de saída

CSRX e CSRY serão atualizados.

Alterações

O registro AF e as locações da memória CSRX e CSRY serão alterados.

FNKSB (*Function key on*) &HØØC9
CALL &HØØC9

Esta chamada verifica se o display de teclas de funções está ativo e imprime suas funções.

Parâmetros de entrada

Se CNSDFG contiver Ø, esta rotina não executará nada; de outra forma, ela seguirá para DSPFNK.

Parâmetros de saída

Não há.

Alterações
Os registros AF, BC e DE podem ser alterados.

ERAFNK (*Erase function key*) &HØØCC
CALL &HØØCC

Esta chamada apaga o display das teclas de função, desde que o VDP esteja em um modo texto.

Parâmetros de entrada

Não há.

Parâmetros de saída

Não há.

Alterações

Os registros AF, BC e DE podem ser alterados por esta chamada.

DSPFNK (*Display function keys*) &HØØCF
CALL &HØØCF

Esta chamada mostra as definições das teclas de função na parte inferior da tela.

Parâmetros de entrada

Não há.

Parâmetros de saída

A locação da memória CNSDFG armazena o valor 255, mostrando que o display de teclas de funções está ativo.

Alterações

Os registros AF, BC e DE e a locação da memória CNSDFG podem ser alterados por esta rotina.

TOTEXT (*To text mode*) &HØØD2
CALL &HØØD2

Esta rotina força a tela a entrar em um modo texto, se ja não estiver.

Parâmetros de entrada

O modo texto que será aceito é armazenado na locação da memória OLDSCR.

Parâmetros de saída

Não há.

Alterações
Esta chamada pode alterar os registros AF, BC, DE e HL e as locações da memória LINLEN, NAMBAS, CGPBAS e ASCRMOD.

CHGCAP (*Change CAPS*) &HØ132
CALL &HØ132

Esta rotina controla o estado do LED indicativo de CAPS LOCK.

Parâmetros de entrada

Se o acumulador contiver Ø, o LED permanecerá apagado.

Parâmetros de saída

Não há.

Alterações

Apenas o acumulador e as flags são alterados por esta chamada.

SNSMAT (*Scan matrix*) &HØ141
CALL &HØ141

Esta chamada executa uma leitura de uma linha da matriz do teclado, e retorna o estado das teclas daquela linha.

Parâmetros de entrada

O acumulador contém o número da linha a ser lida.

Parâmetros de saída

O estado das teclas daquela linha é retornado no acumulador. Se uma dessas teclas for pressionada, o bit correspondente do acumulador será resetado.

Alterações

Apenas o acumulador e as flags são afetados por esta chamada.

ISFLIO &HØ14A
CALL &HØ14A

Esta chamada verifica se algum dispositivo de entrada/saída está ativo.

Parâmetros de entrada

Não há.

Parâmetros de saída

Se não houver dispositivo ativo, o acumulador conterá Ø e a flag Zero será setada.

Alterações

O acumulador e as flags são alterados.

OUTDLP (*Out display printer*) &HØ14D
CALL &HØ14D

Esta rotina é utilizada pelo interpretador Basic para escrever um caractere na impressora.

Parâmetros de entrada

O acumulador deve conter o código do caractere a ser impresso.

Parâmetros de saída

Não há.

Alterações

Apenas as flags são afetadas por esta chamada.

KILBUF (*Kill buffer*) &HØ156
CALL &HØ156

Esta chamada limpa o buffer de teclado.

Parâmetros de entrada

Não há.

Parâmetros de saída

Não há.

Alterações

Apenas o registro HL é alterado.

CAPÍTULO 31 - ROM BIOS QUE CONTROLAM AS PORTAS DOS JOYSTICKS

Os pontos de entrada descritos neste capítulo permitem controle total sobre as portas dos joysticks, bem como sobre todos os dispositivos que a elas possam ser conectados.

GTSTCK (*Get stick*) &HØØD5
CALL &HØØD5

Esta rotina retorna o estado das teclas cursoras ou um dos joysticks.

Parâmetros de entrada

O acumulador deve conter um identificador de joystick, que vale Ø para as teclas cursoras, 1 para o joystick 1 e 2 para o joystick 2.

Parâmetros de saída

A direção selecionada através das teclas cursoras ou de um joystick é armazenada no acumulador. Estas posições são:

Ø = joystick centralizado
1 = para cima
2 = para cima e para a direita
3 = para a direita
4 = para baixo e para a direita
5 = para baixo
6 = para baixo e para a esquerda
7 = para a esquerda
8 = para cima e para a esquerda

Alterações

Os registros AF, BC, DE e HL podem ser modificados por esta chamada.

GTTRIG (*Get trigger*) &HØØD8
CALL &HØØD8

Esta chamada retorna o estado corrente tanto da barra de espaço, quanto de um dos botões de tiro de um joystick.

Parâmetros de entrada

O acumulador especifica qual botão de tiro deve ser lido, conforme valores abaixo:

Ø = barra de espaço
1 = botão 1A
2 = botão 2A
3 = botão 1B
4 = botão 2B

Parâmetros de saída

Se o correspondente botão de tiro estiver pressionado, o acumulador retorna o valor 255.

Alterações

Os registros AF, BC, DE e HL podem ser alterados pela chamada.

GTPAD (*Get pad - touch pad*) &HØØDB
CALL &HØØDB

Esta chamada retorna o estado de uma *Touch pad* conectada a uma das portas de joystick.

Parâmetros de entrada

O registro A deve conter um número na faixa de Ø a 7, dependendo da informação requerida.

Para A na faixa de Ø a 3, é retornada uma informação sobre uma *touch pad* conectada à porta 1 de joystick.

Para A entre 4 e 7, valem os parâmetros de entrada para a porta 2.

Para saber se uma *touch pad* está sendo pressionada, utilizam-se os valores Ø e 4.

Para encontrar a coordenada X do ponto pressionado, utiliza-se 1 ou 5, e 2 ou 6 para a coordenada Y do ponto.

Parâmetros de saída

Estes parâmetros de saída são armazenados no acumulador e dependem dos parâmetros de entrada, da seguinte maneira: no caso de Ø ou 4 na entrada, o acumulador armazenará 255 se a *touch pad* não estiver sendo

pressionada. O mesmo acontece quando um parâmetro de entrada vale 3 ou 7. Se os parâmetros de entrada forem 1 ou 2, as coordenadas X e Y serão armazenadas (na faixa de Ø a 255).

Alterações

Os registros AF, BC, DE e HL podem ser alterados pela chamada.

NOTA: As interrupções são habilitadas por esta rotina. Veja no seu manual, informações sobre a função PAD em Basic para outros detalhes.

GTPDL (*Get paddle*) &HØØDE
CALL &HØØDE

Esta rotina retorna o estado de um dos 12 *paddles* possíveis de serem conectados às portas dos joysticks.

Parâmetros de entrada

O número do *paddle* deve ser armazenado no acumulador, sendo ímpar para um *paddle* conectado na porta 1 e par, se conectado na porta 2.

Parâmetros de saída

Um número na faixa de Ø a 255 é armazenado no acumulador, especificando o estado daquele *paddle*.

Alterações

Os registros AF, BC, DE e Hl podem ser alterados.

CAPÍTULO 32 — ROM BIOS ASSOCIADOS AO CASSETE

As rotinas descritas neste capítulo são utilizadas para controlar a interface do cassete e sistemas de arquivos.

TAPION (*Tape input on*) &HØØE1
CALL &HØØE1

Esta rotina liga o motor do gravador cassete e lê o cabeçalho (*header*) recebido da fita.

Parâmetros de entrada

Não há.

Parâmetros de saída

A flag de transporte será setada se a operação for interrompida.

Alterações

Esta chamada pode alterar os registros AF, BC, DE e HL.

TAPIN (*Tape in*) &HØØE4
CALL &HØØE4

Esta chamada lê um byte recebido da fita cassete.

Parâmetros de entrada

Não há.

Parâmetros de saída

O dado é armazenado no acumulador. A flag de transporte será setada se a operação for interrompida.

Alterações

Os registros AF, BC, DE e HL podem ser alterados por esta rotina.

TAPIOF (*Tape input off*) &HØØE7
CALL &HØØE7

Esta chamada interrompe a leitura da fita cassete.

Parâmetros de entrada

Não há.

Parâmetros de saída

Não há.

Alterações

Não há.

TAPOON (*Tape output on*) &HØØEA
CALL &HØØEA

Esta rotina liga o motor do gravador cassete e escreve um bloco de cabeçalho na fita cassete.

Parâmetros de entrada

O acumulador deve conter Ø se se deseja enviar um header curto.

Parâmetros de saída

A flag de transporte será setada se a operação for interrompida.

Alterações

Os registros AF, BC, DE e HL podem ser modificados por esta rotina.

TAPOUT (*Tape output*) &HØØED
CALL &HØØED

Esta chamada envia um byte para o gravador cassete.

Parâmetros de entrada

O registro A deve conter o byte a ser enviado.

Parâmetros de saída

Se a operação for interrompida, a flag de transporte será setada.

Alterações

Esta rotina pode alterar os registros AF, BC, DE e HL.

TAPOOF (*Tape output off*) &HØØFØ
CALL &HØØFØ

Esta rotina interrompe o envio de bytes para o gravador cassete.

Parâmetros de entrada

Não há.

Parâmetros de saída

Não há.

Alterações

Não há.

STMOTR (*Stop motor*) &HØØF3
CALL &HØØF3

Esta chamada liga ou desliga o motor do gravador cassete, ou muda para o estado oposto.

Parâmetros de entrada

Se o registro A contiver 1 o motor do cassete será ligado; se contiver Ø, o motor será desligado, e se contiver 255, o motor do cassete será invertido, isto é, se estiver ligado, será desligado, e vice-versa.

Parâmetros de saída

Não há.

Alterações

Somente o acumulador e as flags serão alterados por esta rotina.

CAPÍTULO 33 — ROM BIOS QUE TRATAM DO SOM

Os pontos de entrada deste capítulo são usados para controlar o PSG.

GICINI &H0090
CALL &H0090

Esta rotina inicializa o Gerador de Sons Programável.

Parâmetros de entrada

Não há.

Parâmetros de saída

Não há.

Alterações

As locações da memória MUSICF, PLYCNY, VCBA a VCBA+4,

VCBB a VCBB+4 e VCBC a VCBC+4 são zeradas.

WRTPSG (*Write PSG*) &H0093
CALL &H0093

Esta chamada envia um valor para algum registro do PSG.

Parâmetros de entrada

O registro A deve armazenar o número do registro do PSG que irá receber o dado, na faixa de 0 a 13. O byte a ser enviado deve estar armazenado no registro E.

Parâmetros de saída

Não há.

Alterações

Não há.

RDPSG (*Read PSG*) &H0096
CALL &H0096

Esta rotina lê o conteúdo de algum registro do PSG.

Parâmetros de entrada

O acumulador deve conter o número do registro do PSG, na faixa de 0 a 13.

Parâmetros de saída

O conteúdo do registro do PSG é retornado ao acumulador.

Alterações

Somente o acumulador é afetado pela chamada.

STRTMS (*Start music*) &HØØ99
CALL &HØØ99

Esta rotina inicia a execução de uma música se assim o for requerido.

Parâmetros de entrada

Não há.

Parâmetros de saída

Se o buffer de som estiver vazio, o acumulador será resetado.

Alterações

Os registros AF e HL e as locações da memória PLYCNT e MUSICF podem ser alterados por esta rotina.

CAPÍTULO 34 — ROM BIOS ASSOCIADOS AO VDP

Os pontos de entrada no BIOS descritos neste capítulo proporcionam controle completo sobre o VDP do MSX.

DISSCR (*Disable screen*) &HØØ41

CALL &HØØ41

Esta rotina, que "desabilita a tela", quando chamada, limpa a tela, colocando-a com a mesma cor da borda.
Todas as saídas que se fizerem serão enviadas para a tela, mas não serão visíveis, a menos que ENASCR (&HØØ44) seja chamada.

Outra maneira de tornar visível o que for impresso é mudar o modo da tela.

Parâmetros de entrada

Não há.

Parâmetros de saída

Não há.

Alterações

Esta rotina modifica o conteúdo dos registros AF e BC, além de habilitar as interrupções.

NOTA: Esta rotina é para ser utilizada em Basic, por meio da função USR. Ela pode ser usada com a tela desabilitada, para imprimir um desenho, após ser habilitada novamente, dando a impressão de plotagem instantânea.

Por exemplo:

```
10 DEFUSRØ=H41
20 DEFUSR1=H44
30 CLS
40 REM desabilita a tela
50 X=USR(Ø)
60 TU$=INPUT$(1)
70 REM habilita a tela
80 X=USR1(Ø)
```

ENASCR (*Enable screen*) &HØØ44
CALL &HØØ44

Este endereço é chamado para habilitar a tela, após ter sido desabilitada por DISSCR.

Parâmetros de entrada

Não há.

Parâmetros de saída

Não há.

Alterações

Além de habilitar as interrupções, os registros AF e BC são modificados.

WRTVDP (*Write VDP*) &HØØ47
CALL &HØØ47

Este endereço envia um byte para um registro do VDP.

Parâmetros de entrada

O número do registro do VDP a ser acessado deve estar armazenado no registro C, e o dado a ser enviado, no registro B.

Parâmetros de saída

Não há, mas a variável de sistema RGOSAV+C contém o valor inicialmente especificado no registro B.

Alterações

Os registros AF e BC, e a locação da memória RGxSAV são alterados por esta rotina , onde x é o registro onde o dado estava armazenado.

Por exemplo, se o registro C contiver 5 e o registro B armazenar Ø, na saída da rotina a variável RG5SAV será Ø.

NOTA: Possíveis usos:

Esta rotina é muito poderosa, permitindo controle sobre o VDP. Recomenda-se utilizar esta rotina para enviar dados aos registros do VDP, já que são feitas automaticamente cópias de seus conteúdos.

Quando um valor é armazenado num desses registros, não há meio de se saber o seu conteúdo, já que eles são apenas de escrita.

RDVDP (*Read VDP*) &HØ13E
CALL &HØ13E

Esta rotina é usada para ler o estado dos registros do VDP.

Parâmetros de entrada

Não há.

Parâmetros de saída

O acumulador conterá uma cópia do conteúdo do estado do registro do VDP.

Alterações

Somente o acumulador é afetado por este registro.

RDVRM (*Read VRAM*) &HØØ4A
CALL &HØØ4A

Este ponto de entrada lê uma locação da memória VRAM.

Parâmetros de entrada

O par de registros HL deve conter o endereço a ser acessado da VRAM.

Parâmetros de saída

O registro A armazenará o conteúdo do endereço da VRAM

apontado por HL. O estado das flags, na saída desta rotina não refletem o conteúdo do acumulador.

Alterações

O registro A e as flags são modificados por esta rotina.

WRTVRM (*Write VRAM*) &HØØ4D
CALL &HØØ4D

Esta rotina envia o conteúdo do acumulador para um endereço especificado da VRAM.

Parâmetros de entrada

O par de registros HL deve conter o endereço da VRAM que receberá o byte, armazenado no registro A.

Parâmetros de saída

Não há.

Alterações

O acumulador e as flags são afetados por esta rotina, além de habilitar as interrupções.

NOTA: Possíveis usos: acessar a VRAM.

SETRD (*Set to read*) &HØØ5Ø
CALL &HØØ5Ø

Esta chamada ajusta o VDP para uma operação de leitura.

Parâmetros de entrada

O par de registros HL deve armazenar o endereço a ser

lido.

Parâmetros de saída

Não há.

Alterações

Além de habilitar interrupções, esta rotina altera o conteúdo do acumulador, e das flags.

SETWRT (*Set to write*) &H0053
CALL &H0053

Esta rotina ajusta o VDP para uma operação de escrita.

Parâmetros de entrada

O par de registros HL deve armazenar o endereço da VRAM que receberá o dado.

Parâmetros de saída

Não há.

Alterações

Idênticas à anterior.

FILVRM (*Fill VRAM*) &H0056
CALL &H0056

Esta chamada preenche uma área da VRAM, controlada pelo VDP, com um valor constante.

Parâmetros de entrada

O endereço do primeiro byte da área a ser preenchida deve estar armazenado no par de registros HL, o comprimento dessa área deve estar contido no par BC e o código do caractere que será utilizado deve estar armazenado no acumulador.

Parâmetros de saída

Não há.

Alterações

Além de modificar os pares de registros AF e BC, esta rotina habilita as interrupções.

NOTA: Esta rotina é muito útil para preenchimentos de áreas da tela, com uma única cor. É usada pelo comando PAINT.

LDIRMV (*LDIR move*) &H0059
CALL &H0059

Esta rotina move um bloco da memória VRAM para a memória principal.

Parâmetros de entrada

O endereço da VRAM deve estar armazenado no par HL, o comprimento do bloco deve ser especificado no par BC e o endereço-destino da memória principal deve estar contido no par DE.

Parâmetros de saída

O bloco especificado acima, pelos pares de registros DE e BC é retornado pela rotina.

Alterações

Os registros AF, BC e DE são afetados pela rotina, além do bloco em questão.

NOTA: Uma grande utilidade para esta rotina, entre outras, é quando se quer salvar uma tela, transferindo-a para a memória principal e gravando-a. Para recuperá-la, utiliza-se a rotina abaixo.

LDIRVM (*LDIR opposite*) &HØØ5C
CALL &HØØ5C

Em oposição à anterior, esta rotina copia na VRAM um bloco da memória principal.

Parâmetros de entrada

O endereço do bloco deve ser especificado por HL, o endereço destino na VRAM deve estar armazenado no par DE, e o comprimento do bloco no par BC.

Parâmetros de saída

Não há.

Alterações

Esta rotina habilita as interrupções, e modifica o conteúdo dos registros AF, BC e DE.

CHGMOD (*Change Mode*) &HØØ5F
CALL &HØØ5F

Esta chamada ajusta o VDP para um dos seus 4 modos de operação.

Parâmetros de entrada

O registro A deve conter um número equivalente ao modo do VDP, conforme tabela abaixo:

A = Ø -- Modo texto 4Ø colunas
A = 1 -- Modo texto 32 colunas
A = 2 -- Modo gráfico de alta resolução
A = 3 -- Modo multicolorido

Parâmetros de saída

A variável de sistema armazena o número do modo.

Alterações

Os registros alterados são AF, BC, DE e HL. As posições da memória são as variáveis LINLEN, NAMBAS, CGPBAS, SCRMOD e OLDSCR.

CHGCLR (*Change colour*) &HØØ62
CALL &HØØ62

Esta rotina muda as 3 cores da tela se o VDP estiver em modo texto, ou muda a cor da borda se o VDP estiver em modo gráfico.

Parâmetros de entrada

A cor do primeiro plano é lida de FORCLR.
A cor do fundo é lida de BAKCLR.
A cor da borda é lida de CDRCLR.

Parâmetros de saída

Não há.

Alterações

Os registros AF, BC e DE são modificados por esta chamada.

CLRSPR (*Clear sprites*) &HØØ69
CALL &HØØ69

Esta chamada inicializa os sprites. Os padrões são resetados (Ø = transparente), e as cores dos sprites são igualadas com a cor do primeiro plano; as posições verticais são ajustadas para 209 (fora da tela); os nomes dos sprites são igualados com os números dos planos dos sprites, ou seja, o nome do sprite no plano Ø é associado ao código ASCII Ø etc).

Parâmetros de entrada

Não há.

Parâmetros de saída

Não há.

Alterações

Os registros Af, BC, DE e HL são afetados pela rotina.

INITXT (*Initialise text*) &HØØ6C
CALL &HØØ6C

Este ponto de entrada inicializa as posições da memória que descrevem a tela, para modo texto, e então chama SETXT.

Parâmetros de entrada

As variáveis do sistema TXTNAM, TXTCGP e LIN4Ø contêm os parâmetros de entrada.

Parâmetros de saída

SCRMOD = Ø = OLD SCR
NAMBAS = TXTNAM
CGPBAS = TXTCGP
LINLEN = LIN4Ø

ALterações

Os registros AF, BC, DE e HL, e as variáveis RGØSAV, RG1SAV, RG2SAV, RG3SAV, RG4SAV, RG5SAV e RG6SAV, além das variáveis dos parâmetros de saída, são afetados por esta rotina.

INIT32 (*Initialise 32 col. text*) &HØØ6F
CALL &HØØ6F

Esta rotina inicíaliza todas as locações da memória envolvidas com a manipulação da tela, para modo texto 32 colunas, e chama SETT32.

Parâmetros de entrada

As variáveis T32NAM, T32CGP, T32COL, T32ATR, T32PAT e LINL32 contêm os parâmetros de entrada.

Parâmetros de saída

SCRMOD = 1
OLDSCR = 1
ATRBAS = T32ATR
NAMBAS = T32NAM
PATBAS = T32PAT
LINLEN = LIN32

Alterações

Além das variáveis listadas acima, as variáveis de sistema RGØSAV, RG1SAV, RG2SAV, RG3SAV, RG4SAV, RG5SAV e RG6SAV e os registros AF, BC, DE e HL são modificados por esta rotina.

INIGRP (*Initialise graphic mode*)&HØØ72

CALL &HØØ72

Esta rotina inicialisa todas as locações da memória para modo gráfico em alta resolução, e chama SETGRP.

Parâmetros de entrada

As variáveis GRPNAM, GRPCGP, GRPCOL, GRPATR e GRPPAT armazenam os parâmetros de entrada desta chamada.

Parâmetros de saída

SCRMOD = 2, PATBAS = GRPPAT E ATRBAS = GRPATR

Alterações

Os registros AF, BC, DE e HL, as variáveis RGØSAV, RG1SAV, RG2SAV, RG3SAV, RG4SAV, RG5SAV e RG6SAV, além das variáveis dos parâmetros de saída são alterados por esta rotina.

INIMLT (***Initialise multicolour mode***) &HØØ75
CALL &HØØ75

Esta chamada inicializa todas as locações da memória para modo multicolorido, e chama SETMLT.

Parâmetros de entrada

As variáveis MLTNAM, MLTCGP, MLTCOL, MLTATR e MLTPAT armazenam os parâmetros de entrada.

Parâmetros de saída

SCRMOD = 3, PATBAS = MLTPAT e ATRBAS = MLTATR

Alterações

Idênticas à anterior.

SETTXT (*Set text mode*) &HØØ78
CALL &HØØ78

Esta rotina ajusta os registros do VDP para modo texto 4Ø colunas.

Parâmetros de entrada

Não há.

Parâmetros de saída

Não há.

Alterações

Os registros AF, BC, DE e HL e as variáveis RGØSAV, RG1SAV, RG2SAV, RG3SAV, RG4SAV, RG5SAV e RG6SAV são modificados por esta rotina.

SETT32 (*Set text 32 col*) &HØØ7B
CALL &HØØ7B

Esta rotina ajusta os registros do VDP para modo texto de 32 colunas.

Parâmetros de entrada

Não há.

Parâmetros de saída

Não há.

Alterações

Idênticas à anterior

SETGRP (*Set graphic mode*) &HØØ7E
CALL &HØØ7E

Esta rotina ajusta os registros do VDP para modo gráfico de alta resolução.

Parâmetros de entrada

Não há.

Parâmetros de saída

Não há.

Alterações

As mesmas que a chamada anterior.

SETMLT (*Set multicolour mode*) &HØØ81
CALL &HØØ81

Esta rotina ajusta os registros do VDP para modo multicolorido.

Parâmetros de entrada

Não há.

Parâmetros de saída

Não há.

Alterações

Idem.

CALPAT (*Call pattern*) &H0084
CALL &H0084

Esta rotina retorna o endereço do padrão de um sprite na VRAM.

Parâmetros de entrada

O número do sprite, entre 0 e 31 deve estar armazenado no registro A.

Parâmetros de saída

O endereço do padrão do sprite retorna no par de registros HL.

Alterações

Os registros AF, DE e HL são afetados por esta rotina.

CALATR (*Call attribute*) &H0087
CALL &H0087

Esta rotina retorna o endereço da tabela de atributos de um sprite na VRAM.

Parâmetros de entrada

O número do sprite deve estar armazenado no acumulador.

Parâmetros de saída

O registro HL armazena o endereço da tabela de atributos daquele sprite.

Alterações

Os registros DE e HL e as flags são modificados por esta rotina.

GSPSIZ (*Graphic sprite size*) &HØØ8A
CALL &HØØ8A

Esta chamada retorna o tamanho do sprite corrente em termos de número de bytes ocupados por cada sprite (8 ou 32).

Parâmetros de entrada

Não há.

Parâmetros de saída

O número de bytes por sprite é armazenado no acumulador. Se um sprite 16x16 está em uso, a flag de transporte será setada.

Alterações

O registro A e as flags são modificados por esta rotina.

GRPPRT (*Graphics print*) &HØØ8D
CALL &HØØ8D

Esta rotina é usada para imprimir um caractere na tela gráfica de alta resolução, na posição do cursor gráfico.

Parâmetros de entrada

O código do caractere a ser impresso deve estar armazenado no acumulador.

Parâmetros de saída

Não há.

Alterações

Não há.

SCALXY (*Scale X/Y*) &HØ1ØE
CALL &HØ1ØE

Esta rotina assegura que um ponto, gerado nos registros, permaneça na tela. Se não ocorrer, as suas coordenadas serão truncadas e será emitida uma mensagem de erro. Truncar significa que as coordenadas são muito grandes; elas são comparadas com o valor máximo permitido, e se resultarem negativas, serão zeradas. Se o VDP estiver em modo multicolorido, as coordenadas X e Y serão divididas por 4, já que este modo possui 64 pontos na horizontal e 48 pontos na vertical.

Parâmetros de entrada

O par de registros BC deve armazenar a coordenada X, e o par DE, a coordenada Y.

Parâmetros de saída

Na saída, os registros BC e DE contêm as coordenadas truncadas X e Y, respectivamente. Se ambas estiverem fora da faixa permitida, a flag de transporte será resetada.

Alterações

Os registros AF, BC e DE são modificados por esta rotina.

MAPXYC (*Map XY coord.*) &HØ111
CALL &HØ111

Esta rotina, muito útil por sinal, calcula o endereço na VRAM de um pixel, tanto em modo gráfico de alta resolução, quanto em modo multicolorido. Também retorna a posição no byte que representa o pixel.

Parâmetros de entrada

As coordenadas do ponto em questão devem estar armazenadas nos pares de registros BC (X) e DE (Y).

Esta rotina utiliza SCRMOD para determinar qual o modo da tela, GRPCGP para encontrar o endereço inicial da tabela de padrões de gráficos de alta resolução e MLTCGP para determinar o endereço inicial da tabela de padrões do modo multicolorido.

Parâmetros de saída

O endereço da VRAM, do byte contendo o pixel, é retornado na variável CLOC, e o byte descritivo do pixel é retornado em CMASK. No modo gráfico de alta resolução, cada pixel é representado por um bit (primeiro plano = 1 e fundo = Ø). Este bit está exatamente na mesma posição que o bit setado em CMASK, ou seja, se o pixel está armazenado no bit 5 do byte apontado por CLOC , então CMASK conterá ØØ1ØØØØ. No modo multicolorido, cad pixel é representado por 4 bits - a cor do pixel, e novamente a posição daqueles 4 bits correspondentes aos bits setados em CMASK, ou seja, se o pixel está armazenado nos bits Ø a 3 do byte apontado, então CMASK conterá ØØØØ1111.

Alterações

Os registros AF, D e HL e as variáveis de sistema CLOC e CMASK são modificados por esta rotina.

FETCHC (***Fetch current pixel***) &HØ114
CALL &HØ114

Esta rotina simplesmente lê o endereço do pixel corrente e sua máscara (padrão).

Parâmetros de entrada

A variável CLOC deve armazenar o endereço do pixel e a variável CMASK deve conter sua máscara (padrão).

Parâmetros de saída

O par HL armazenará o conteúdo de CLOC, e o registro A conterá o valor armazenado em CMASK.

Alterações

Os registros HL e AF são alterados pela rotina.

STOREC (***Store current***) &HØ117
CALL &HØ117

Esta chamada simplesmente armazena o endereço do pixel e sua respectiva máscara na memória.

Parâmetros de entrada

O par HL deve conter o endereço corrente, e o registro A, o valor correspondente à sua máscara.

Parâmetros de saída

CMASK conterá uma cópia do conteúdo do acumulador, e CLOC conterá o valor armazenado em HL.

Alterações

As variáveis CMASK e CLOC são afetadas pela rotina.

SETATR (*Set attributes*) &HØ11A
CALL &HØ11A

Esta rotina iguala o byte de atributos ao conteúdo do acumulador, se o número contido no acumulador for menor que 16; de outra forma, o byte de atributos não é alterado.

Parâmetros de entrada

O acumulador deve conter o valor que será enviado ao byte de atributos.

Parâmetros de saída

A variável ATRBYT conterá o valor especificado no acumulador, desde que seja menor que 16.

Alterações

A variável ATRBYT pode ser modificada pela rotina.

READC (*Read colour*) &HØ11D
CALL &HØ11D

Esta rotina lê o atributo (cor) do pixel corrente.

Parâmetros de entrada

O endereço do pixel em questão, bem como sua máscara, devem estar armazenados em CLOC e CMASK, respectivamente.

Parâmetros de saída

A cor do pixel será armazenada no acumulador.

Alterações

Somente o acumulador e as flags são afetados por esta rotina.

SETC (*Set colour*) &HØ12Ø
CALL &HØ12Ø

Esta chamada colore o pixel com a cor especificada. Se o VDP estiver em modo gráfico de alta resolução, e se a cor especificada não for a mesma que a cor do primeiro plano e a cor de fundo, então a cor de fundo de todos os pixéis contidos no byte que armazena o pixel corrente será trocada pela cor especificada.

Parâmetros de entrada

O endereço do pixel corrente bem como sua máscara devem estar contidos em CLOC e CMASK respectivamente, e a cor que será utilizada pelo pixel deve ser especificada por ATRBYT.

Parâmetros de saída

Não há.

Alterações

Somente o acumulador e as flags são modificados pela rotina.

RIGHTC (*Right cursor*) &HØØFC
CALL &HØØFC

Esta rotina move o cursor gráfico uma posição à direita.

Parâmetros de entrada

O endereço do byte contendo o pixel é armazenado na variável CLOC e a sua máscara é armazenada em CMASK.

Parâmetros de saída

Os conteúdos dessas duas variáveis são atualizados em função da nova posição do cursor.

Alterações

Os registros Af e as duas variáveis citadas acima podem ser alterados por esta rotina.

LEFTC (*Left cursor*) &HØØFF
CALL &HØØFF

Esta rotina move o cursor gráfico uma posição para a esquerda.

Parâmetros de entrada

Idem.

Parâmetros de saída

Idem.

Alterações

Idem.

UPC (*Up cursor*) &HØ1Ø2
CALL &HØ1Ø2

Esta rotina move o cursor gráfico uma posição para cima, desde que não seja o topo da tela, pois, neste caso, sua posição permanecerá inalterada.

Parâmetros de entrada

Idem.

Parâmetros de saída

Idem.

Alterações

Idem.

TUPC (*Top up cursor*) &HØ1Ø5
CALL &HØ1Ø5

Esta rotina move o cursor gráfico uma posição para cima, desde que não seja o topo da tela, pois, neste caso, sua posição permanecerá inalterada e a flag de transporte será setada.

Parâmetros de entrada

Idem.

Parâmetros de saída

Idem.

Alterações

Idem.

DOWNC (*Down cursor*) &HØ1Ø8

CALL &HØ1Ø8

Esta rotina move o cursor gráfico uma posição para baixo, desde que não ultrapasse o limite inferior da tela, pois, neste caso, a sua posição permanecerá inalterada.

Parâmetros de entrada

Idem.

Parâmetros de saída

Idem.

Alterações

Idem.

TDOWNC (*Top down cursor*) &HØ1ØB
CALL &HØ1ØB

Esta rotina move o cursor gráfico uma posição para baixo, desde que não ultrapasse o limite inferior da tela, pois, neste caso, sua posição permanecerá inalterada e a flag de transporte será setada.

Parâmetros de entrada

Idem.

Parâmetros de saída

Idem.

Alterações

Idem.

NSETCX (*Number set coordin. X*) &HØ123
CALL &HØ123

Esta chamada seta um número especificado de pixéis à direita do cursor gráfico, com uma cor dada.

Parâmetros de entrada

O par HL deve conter o número de pixéis a serem coloridos, a variável de sistema ATRBYT deve conter a cor dos pixéis, e os endereços do cursor gráfico e sua máscara devem estar armazenados nas variáveis CLOC e CMASK, respectivamente.

Parâmetros de saída

Não há.

Alterações

Os registros AF, BC, DE e HL e as variáveis CLOC e CMASK podem ser alterados por esta rotina.

GTASPC (*Get aspect circle*) &HØ126
CALL &HØ126

Esta rotina é utilizada pelo interpretador Basic para o comando CIRCLE, com o propósito de encontrar os parâmetros correntes do comando.

Parâmetros de entrada

A variável ASPCT1 deve armazenar 256/(raio) e ASPCT2 deve conter 256*(raio).

Parâmetros de saída

O par de registros HL armazenará o conteúdo de ASPCT2 e o par DE, ASPCT1.

Alterações

Somente os registros HL e DE são modificados pela rotina.

PNTINI (*Paint initialise*) &HØ129
CALL &HØ129

Esta é a rotina de inicialização do comando PAINT do Basic.

Parâmetros de entrada

O acumulador deve armazenar a cor que será utilizada.

Parâmetros e saída

Se o VDP estiver em modo multicolorido, a variável BRDATR conterá uma cópia do parâmetro de entrada, e, de outra forma, conterá uma cópia do valor armazenado em ATRBYT.

Alterações

O acumulador, as flags e a variável de sistema BRDATR podem ser alterados pela rotina.

SCANR (*Scan pixel right*) &HØ12C
CALL &HØ12C

Esta rotina lê um número especificado de pixéis à direita do cursor gráfico, colorindo-os com uma

determidada cor, até que encontre um pixel com a cor da borda, ou atinja o limite da tela, ou complete o número especificado.

Parâmetros de entrada

O número máximo de pixéis a serem lidos e coloridos deve estar armazenado no par DE (até 256), a cor da borda deve estar contida na variável BRDATR, e a cor com a qual serão coloridos os pixéis deve estar em ATRBYT.

Parâmetros de saída

Não há.

Alterações

Os registros AF, BC, DE e HL e as variáveis de sistema CLOC, CMASK e FILNAM até FILNAM+3 podem ser modificados pela rotina.

SCANL (*Scan pixel left*) &HØ12F
CALL &HØ12F

Esta rotina executa tarefa idêntica à anterior, só que para a esquerda.

Parâmetros de entrada

Idem.

Parâmetros de saída

Não há.

Alterações

Idem.

CAPÍTULO 35 — TABELAS DO BIOS REFERENTES AO TECLADO

No endereço &HØD89 da ROM BIOS existe uma rotina que converte cada bit ativo de um byte de transição de uma linha de teclado em um código de uma tecla. O bit primeiramente é convertido no número da tecla, determinado pela sua posição na matriz do teclado:

7	6	5	4	3	2	1	Ø	Linha
7 Ø7H	6 Ø6H	5 Ø5H	4 Ø4H	3 Ø3H	2 Ø2H	1 Ø1H	Ø ØØH	Ø
; ØFH	] ØEH	[ØDH	\ ØCH	= ØBH	- ØAH	9 Ø9H	8 Ø8H	1
B 17H	A 16H	libr 15H	/ 14H	. 13H	, 12H	11H	1ØH	2
J 1FH	I 1EH	H 1DH	G 1CH	F 1BH	E 1AH	D 19H	C 18H	3
R 27H	Q 26H	P 25H	O 24H	N 23H	M 22H	L 21H	K 2ØH	4
Z 2FH	Y 2EH	X 2DH	W 2CH	V 2BH	U 2AH	T 29H	S 28H	5
F3 37H	F2 36H	F1 35H	CODE 34H	CAP 33H	GRAP 32H	CTRL 31H	SHIF 3ØH	6
CR 3FH	SEL 3EH	BS 3DH	STOP 3CH	TAB 3BH	ESC 3AH	F5 39H	F4 38H	7
RIGH 47H	DOWN 46H	UP 45H	LEFT 44H	DEL 43H	INS 42H	HOME 41H	SPAC 4ØH	8
4 4FH	3 4EH	2 4DH	1 4CH	Ø 4BH	4AH	49H	48H	9
. 57H	, 56H	- 55H	9 54H	8 53H	7 52H	6 51H	5 5ØH	1Ø

Colunas

No endereço &HØDA5 existe uma tabela que contém os códigos dos números das teclas de &HØØ a &H2F, para várias combinações das teclas de controle.

Um zero como entrada na tabela significa que nenhum código de tecla será produzido quando alguma tecla for pressionada:

									Lin.
	37H	36H	35H	34H	33H	32H	31H	3ØH	Ø
	3BH	5DH	5BH	5CH	3DH	2DH	39H	38H	1
NORMAL	62H	61H	9CH	2FH	2EH	2CH	6ØH	27H	2
	6AH	69H	68H	67H	66H	65H	64H	63H	3
	72H	71H	7ØH	6FH	6EH	6DH	6CH	6BH	4
	7AH	79H	78H	77H	76H	75H	74H	73H	5
	26H	5EH	25H	24H	23H	4ØH	21H	29H	Ø
	3AH	7DH	7BH	7CH	2BH	5FH	28H	2AH	1
SHIFT	42H	41H	9CH	3FH	3EH	3CH	7EH	22H	2
	4AH	49H	48H	47H	46H	45H	44H	43H	3
	52H	51H	5ØH	4FH	4EH	4DH	4CH	4BH	4
	5AH	59H	58H	57H	56H	55H	54H	53H	5
	FBH	F4H	BDH	EFH	BAH	ABH	ACH	Ø9H	Ø
	Ø6H	ØDH	Ø1H	1EH	F1H	17H	Ø7H	ECH	1
GRAPH	11H	C4H	9CH	1DH	F2H	F3H	BBH	Ø5H	2
	C6H	DCH	13H	15H	14H	CDH	C7H	BCH	3
	18H	CCH	DBH	C2H	1BH	ØBH	C8H	DDH	4
	ØFH	19H	1CH	CFH	1AH	CØH	12H	D2H	5
	ØØH	F5H	ØØH	ØØH	FCH	FDH	ØØH	ØAH	Ø
	Ø4H	ØEH	Ø2H	16H	FØH	1FH	Ø8H	ØØH	1
SHIFT	ØØH	FEH	9CH	F6H	AFH	AEH	F7H	Ø3H	2
GRAPH	CAH	DFH	D6H	1ØH	D4H	CEH	C1H	FAH	3
	A9H	CBH	D7H	C3H	D3H	ØCH	C9H	DEH	4
	F8H	AAH	F9H	DØH	D5H	C5H	ØØH	D1H	5

	7	6	5	4	3	2	1	Ø	
	E1H	EØH	98H	9BH	BFH	D9H	9FH	EBH	Ø
	B7H	DAH	EDH	9CH	E9H	EEH	87H	E7H	1
CODE	97H	84H	9CH	A7H	A6H	86H	E5H	B9H	2
	91H	A1H	B1H	81H	94H	8CH	8BH	8DH	3
	93H	83H	A3H	A2H	A4H	E6H	B5H	B3H	4
	95H	AØH	8AH	88H	95H	82H	96H	89H	5
	ØØH	ØØH	9DH	9CH	BEH	9EH	ADH	D8H	Ø
	B6H	EAH	E8H	ØØH	ØØH	ØØH	8ØH	E2H	1
SHIFT	ØØH	8EH	9CH	A8H	ØØH	8FH	E4H	B8H	2
CODE	92H	ØØH	BØH	9AH	99H	ØØH	ØØH	ØØH	3
	ØØH	ØØH	E3H	ØØH	A5H	ØØH	B4H	B2H	4
	ØØH	ØØH	ØØH	ØØH	ØØH	9ØH	ØØH	ØØH	5

Colunas

No endereço &H1Ø33 existe outra tabela que contém os códigos das teclas de números &H3Ø a &H57.

7	6	5	4	3	2	1	Ø	Lin.
ØØH	ØØH	ØØH	ØØH	ØØH	ØØH	ØØH	ØØH	6
ØDH	18H	Ø8H	ØØH	Ø9H	1BH	ØØH	ØØH	7
1CH	1FH	1EH	1DH	7FH	12H	ØCH	2ØH	8
34H	33H	32H	31H	3ØH	ØØH	ØØH	ØØH	9
2EH	2CH	2DH	39H	38H	37H	36H	35H	1Ø

Colunas

No endereço &H1B97 existe outra tabela, de 2Ø bytes que é utilizada pelo decodificador de teclado para encontrar a rotina relacionada com um número de uma tecla:

Número da Tecla	Endereço	Função
ØØH A 2FH	&HØF83	Linhas Ø a 5

3ØH A 32H	&HØF1Ø	SHIFT, CTRL, GRAPH
33H	&HØF36	CAP
34H	&HØF1Ø	CODE
35H A 39H	&HØFC3	F1 A F5
3AH A 3BH	&HØF1Ø	ESC, TAB
3CH	&HØF46	STOP
3DH A 4ØH	&HØF1Ø	BS, CR, SEL, SPACE
41H	&HØFØ6	HOME
42H A 57H	&HØF1Ø	INS, DEL, CURSOR

CAPÍTULO 36 - PRINCIPAIS ROTINAS DO INTERPRETADOR BASIC DA ROM DO MSX

Neste capítulo daremos resumidamente os endereços iniciais das principais rotinas do Interpretador Basic, com os seus respectivos significados.

268C - Rotina utilizada pelo avaliador de expressões da ROM para subtrair dois valores de dupla precisão.

269A - Rotina utilizada pelo avaliador de expressões para somar dois valores em dupla precisão.

27E6 - Rotina utilizada pelo avaliador de expressões que multiplica dois valores de dupla precisão.

289F - Divisão de dois valores de dupla precisão.

2993 - Aplicação da função COS em um operando de dupla precisão.

29AC - Função SIN em um operando de dupla precisão.

29FB - Função TAN.

2A14 - Função ATN.

2A72 - Função LOG.

2AFF - Função SQR.

2B4A - Função EXP.

2BDF - Função RND.

2E82 - Função ABS.

2E97 - Função SGN.

2F21 - Rotina usada para encontrar a relação (menor que, maior que ou igual) entre dois valores de simples precisão.

2F4D - Rotina usada para encontrar a relação (maior que, menor que ou igual) entre dois valores inteiros.

2F83 - Rotina usada para encontrar a relação (maior que, menor que ou igual) entre dois valores de dupla precisão.

2FB2 - Função CSNG.

303A - Função CDBL.

30BE - Função FIX.

30CF - Função INT.

3167 - Subtração de dois operandos inteiros.

3172 - Adição de dois operandos inteiros.

3193 - Multiplicação de dois operandos inteiros.

31E6 - Divisão de dois operandos inteiros.

324E - Adição de dois valores de simples precisão.

3257 - Subtração de dois valores de simples precisão.

325C - Multiplicação de dois valores de simples precisão.

3265 - Divisão de dois valores de simples precisão.

37C8 - Exponenciação entre dois valores de simples precisão.

37D7 - Exponenciação entre dois valores de dupla precisão.

383F - Exponenciação entre dois valores inteiros.

3D75 - Tabela que contém as mensagens de erro do interpretador.

3FD7 - Mensagem "Ok",CR,LF terminada com um byte de valor Ø.

3FDC - Mensagem "Break" terminada com um byte de valor Ø.

4ØØ1 - Função INP.

4Ø16 - Declaração OUT.

4Ø1C - Declaração WAIT.

4524 - Declaração FOR.

46Ø1 - RUNLOOP - Responsável pela execução de um programa.

4718 - Declaração DEFSTR.

471B - Declaração DEFINT.

471E - Declaração DEFSNG.

4721 - Declaração DEFDBL.

479E - Declaração RUN.

47B2 - Declaração GOSUB.

47E8 - Declaração GOTO.

4821 - Declaração RETURN.

485B - Declaração DATA.

488Ø - Declaração LET.

48E4 - Declaração ON ERROR.

49ØD - Declaração ON DEVICE GOSUB.

4943 - Declaração ON EXPRESSION.

495D - Declaração RESUME.

49AA - Declaração ERROR.

49B5 - Declaração AUTO.

49E5 - Declaração IF.

4A1D - Declaração LPRINT.

4A24 - Declaração PRINT.

4BØE - Declarações LINE INPUT, LINE INPUT$ e LINE.

4B3A - Mensagem "Redo from start", CR, LF e um byte, de valor Ø.

4B62 - Declaração INPUT#.

4B6C - Declaração INPUT.

4B9F - Declaração READ.

4C2F - Mensagem de text"?Extra ignored", CR, LF e um byte de valor Ø.

4C64 - Avaliador de expressões.

4DB8 - Divisão de dois operandos inteiros

4DFD - Função ERR.

4EØB - Função ERL.

4E41 - Função VARPTR.

4F57 - Aplicação de operador relacional (maior, menor ou igual) a um par de operandos.

4F63 - Aplicação do operador lógico NOT.

4F78 - Aplicação de um operador lógico (OR, AND, XOR, EQV, IMP) ou MOD e / a dois operandos numéricos.

4FC7 - Função LPOS.

4FCC - Função POS.

4FD5 - Função USR.

5ØØE - Declaração DEFUSR.

5Ø1D - Declaração DEF FN.

5Ø4Ø - Função FN.

51C9 - Declaração WIDHT.

5229 - Declaração LLIST.

522E - Declaração LIST.

53E2 - Declaração DELETE.

541C - Função PEEK.

5423 - Declaração POKE.

5468 - Declaração RENUM.

555A - Mensagem de texto "Undefined line"

55A8 - Declaração CALL.

57E5 - Declaração PRESET.

57EA - Declaração PSET.

58Ø3 - Função POINT.

58A7 - Declaração LINE.

58BF - Operação de "Boxfill" (BF).

58FC - Desenha uma linha.

593C - Desenha uma linha entre dois pontos, X1 e Y1, armazenados nos registros BC e DE e X2 e Y2, armazenados em GXPOS e GYPOS.

59BC - Geração do erro "Illegal function call" se a tela não estiver em modo gráfico.

59C5 - Declaração PAINT.

5B11 - Declaração CIRCLE.

5D6E - Declaração DRAW.

5E09 - Declaração DIM.

5EA4 - Rotina de busca de variável.

5FBA - Rotina de busca de matriz.

601B - Declaração PRINT USING.

6286 - Declaração NEW.

63C9 - Declaração RESTORE.

63EA - Declaração END.

6424 - Declaração CONT.

6438 - Declaração TRON.

6439 - Declaração TROFF.

643E - Declaração SWAP.

6477 - Declaração ERASE.

64AF - Declaração CLEAR.

6527 - Declaração NEXT.

65C8 - Rotina usada para encontrar a relação (maior, menor ou igual) entre duas strings.

65FA - Função HEX$.

65FF - Função BIN$.

6604 - Função STR$.

6787 - Concatenação entre duas strings.

67FF - Função LEN.

680B - Função ASC.

681B - Função CHR$.

6829 - Função STRING$.

6848 - Função SPACE$.

6861 - Função LEFT$.

6891 - Função RIGHT$.

689A - Função MID$.

68BB - Função VAL.

68EB - Função INSTR.

696E - Declaração MID$.

69F2 - Função FRE.

6AB7 - Declaração OPEN.

6B5B - Declarações LOAD, MERGE e "RUN...

6BA3 - Declaração SAVE.

6C14 - Declaração CLOSE.

6C2A - Declaração LFILES.

6C2F - Declaração FILES.

6C87 - Função INPUT$.

6D03 - Função LOC.

6D14 - Função LOF.

6D25 - Função EOF.

6D39 - Função FPOS.

6E92 - Declaração BSAVE.

6EC6 - Declaração BLOAD.

6FB7 - Declaração CSAVE.

703F - Declarações CLOAD e CLOAD?.

70FF - Mensagem "Found".

7106 - Mensagem "Skip".

7347 - Função INKEY$.

73B7 - Declaração MOTOR.

73CA - Declaração SOUND.

73E5 - Declaração PLAY.

7758 - Declaração PUT.

775B - Declaração GET.

7766 - Declaração LOCATE.

77A5 - Declarações STOP ON/OFF/STOP.

77AB - Declarações SPRITE ON/OFF/STOP.

77B1 - Declarações INTERVALON/OFF/STOP.

77BF - Declarações STRIG ON/OFF/STOP.

786C - Declaração KEY.

78AE - Declarações KEY n, KEY (n) ON/OFF/ STOP, KEY

ON e KEY OFF.

7909A - Função CSRLIN.

7940 - Função STICK.

794C - Função STRIG.

795A - Função PDL.

7969 - Função PAD.

7980 - Declaração COLOR.

79CC - Declaração SCREEN.

7A48 - Declaração SPRITE.

7A84 - Função SPRITE$.

7AAF - Declarações GET/PUT SPRITE.

7B37 - Declaração VDP.

7B5A - Declaração BASE.

7BF5 - Função VPEEK.

7C16 - Declaração DSKØ$.

7C1B - Declaração SET.

7C20 - Declaração NAME.

7C25 - Declaração KILL.

7C2F - Declaração COPY.

7C39 - Função DSKF.

7C3E - Função DSKI$.

7C43 - Função ATTR$.

7C48 - Declaração LSET.

7C4D - Declaração RSET.

7C52 - Declaração FIELD.

7C57 - Função MKI$.

7C5C - Função MKS$.

7C61 - Função MKD$.

7C66 - Função CVI.

7C6B - Função CVS.

7C70 - Função CVD.

7E4B - Declaração MAXFILES.

7ED8 - Mensagem "MSX system".

7EE4 - Mensagem "Versão 1...".

7EF2 - Mensagem "MSX Basic".

7EFD - Mensagem "Copyright...".

7F1B - Mensagem "Bytes free".

CAPÍTULO 37 — ROTINAS EM LINGUAGEM DE MÁQUINA

Infelizmente, devido à absoluta falta de espaço, já que este livro ficou maior do que deveria, apresentarei aqui alguns exemplos de aplicação da linguagem de máquina.

A minha idéia inicial referente a este capítulo era de pegar um programa de 16K, como por exemplo o HYPER RALLY da KONAMI, e abri-lo por inteiro, mostrando todas as suas sub-rotinas, como a de imprimir textos na tela, com os respectivos truques daquela "software house" para "criptografar" as palavras que surgem na tela, ou a rotina que controla o seu combustível, fazendo com que o jogo atinja o seu final.

Este trabalho todo está praticamente pronto, mas infelizmente não coube aqui.

Imagine uma listagem "disassemblada" e comentada, quase que item a item, de cerca de 16000 caracteres - é quase outro livro! Quem sabe...

Vamos lá: a rotina apresentada a seguir serve para fazer com que qualquer programa lido originalmente de um cartucho seja executado a partir de uma fita cassete ou um disquete.

Como já vimos, qualquer programa de cartucho contém como bytes iniciais os códigos "AB", ou seja &H41 e &H42. Imediatamente após esses códigos, vem o seu endereço de execução, na forma byte menos significativo e byte mais significativo, ou seja, &HFF40 quer dizer endereço de execução &H40FF.

Pois bem, digamos que você esteja utilizando, como eu, o programa MON 80, para digitar listagens em códigos hexadecimais.

O seu programa de cartucho será carregado no endereço &H0100, e terminará no endereço &H40FF, totalizando &H4000 ou 16384 bytes, que cabem numa EPROM 27128 (dividindo 128 por 8 você obtém o total de bytes daquele chip).

Portanto o byte &H41 ocupará o endereço &H100, o byte &H42, o endereço &H101, e assim por diante.

Primeiramente, você deverá fazer um "header" para seu programa, para que ele possa ser lido de volta do cassete ou do disquete.

Nesse header, o primeiro byte deve ser um &HFE, que significa bloco de bytes. Em seguida, virão o endereço inicial do programa, o endereço final e o endereço de execução, ou seja, neste header serão digitados 7 bytes. Portanto, o comprimento do programa já passou para &H4007.

Vamos supor que o programa vá rodar no endereço &H8000 (endereço inicial) e, devido ao seu comprimento, vá terminar no endereço &H8000+&H4007, ou seja, &HC007.

Aguarde mais um pouco, pois falta a rotina que vai

transferir o programa das áreas da memória e, conseqüentemente, esse não é seu comprimento total.

A seguir, a rotina que executará essa tarefa, considerando-se que o seu programa está armazenado, com o header que você criou, a partir do endereço &H100:

```
ENDER.   CODIGO   MNEMONICA        COMENTARIOS

4107     DBD0     IN A, (D0)       Lê porta
4109     E680     AND 80           Oper. lógica AND
410B     CA00C0   JP Z, C000       Salto relativo
                                   para endereço
                                   C000, se a página
                                   1 estiver aberta
410E     F3       DI               Desabilita interrupções

410F     DBA8     IN A, (A8)       Lê porta A8 da
                                   PPI
4111     FEF0     CP F0            Comparação lógica
                                   com HF0 para
                                   abrir página 1
4113     2004     JR NZ, LOO       Salto relativo
                                   para LOO
4115     3EFC     LD A, FC         Carrega registro
                                   A com o valor FC
4117     1802     JR INICIO        Salto relativo
                                   para INÍCIO

LOO
4119     3EA8     LD A, A8         Carrega registro
                                   A com o valor A8

INÍCIO
411B     D3A8     OUT (A8),A       Porta A8 da PPI
411D     210080   LD HL,8000       Carrega par de
                                   registros HL com
                                   endereço 8000 ou
                                   origem
4120     110040   LD DE,4000       Carrega par de
                                   registros DE com
```

```
                                endereço 4ØØØ ou
                                destino
4123    Ø1ØØ4Ø  LD BC,4ØØØ      Carrega par de
                                registros BC com
                                valor 4ØØØ, ou
                                seja, comprimen-
                                to do bloco
4126    EDBØ    LDIR            Move bloco na
                                memória
4128    2AØ24Ø  LD HL,(4ØØ2)
                                Carrega par de
                                registros HL com
                                o conteúdo apon-
                                tado pelo endereço
                                4ØØ2
412B    E9      JP (HL)         Salta para o endereço
                                apontado
                                pelo par HL
```

Portanto, a partir do endereço &HØ1ØØ, os códigos devem estar assim armazenados:

```
Ø1ØØ FE ØØ 8Ø 25 CØ ØØ CØ 41 42 FF 4Ø ØØ...
```

Aí está. Um programa armazenado em fita com esta rotina deverá ter os seguintes endereços:

```
INÍCIO:     &H8ØØØ
FINAL:      &HCØ25
EXECUÇÃO:   &HCØØØ
```

O procedimento inverso também é valido, ou seja, fazer um programa armazenado em fita cassete ou disquete virar um programa de cartucho.

Só que o trabalho é um pouco mais extenso e requer mais cuidados.

Logo após o header do programa, você deve inserir os códigos "AB", o endereço de execução e em seguida uma rotina que transfere os bytes do programa, armazenado a

partir do endereço &H4000, por exemplo, para a sua área original.

EDITOR DE CARACTERES

Esta rotina em linguagem de máquina permite que você altere os caracteres padrões do seu MSX, talvez para tornar mais atraentes as letras dele, ou para adequar esses caracteres a uma impressora importada.

Quando o programa é carregado, primeiramente ele faz uma cópia dos 2K do conjunto de caracteres armazenado na ROM para o buffer CHRTAB (&HE2A3 até &HEAA2), e apresenta sua tela inicial, com esses caracteres ampliados.

A rotina possui dois modos de operação: o modo comando e o modo edição, com a tecla RETURN sendo usada para se passar de um modo para outro.

No modo **Comando** as teclas cursoras são utilizadas para seleção do caractere a ser editado, destacado dos outros, por estar sob o cursor intermitente. Do lado direito da tela, os caracteres são mostrados de uma forma ampliada. Pressionando-se a tecla "Q" (maiúscula), o programa retornará ao Basic. Pressionando-se a tecla "A" (maiúscula), far-se-á com que o micro adote aquele conjunto de caracteres, quando então ele será copiado na parte mais alta da memória, de &HEB80 até &HF37F.

No modo de **Edição**, as teclas cursoras são utilizadas para selecionar o pixel a ser editado, marcado por um pequeno cursor, na forma ampliada. A barra de espaços apagará aquele pixel, enquanto que a tecla "." o desenhará. O caractere em questão, mostrado no seu

tamanho normal, será atualizado, conforme as modificações feitas.

O conjunto de caracteres armazenado em CHRTAB pode ser salvo em cassete (não em disco, por causa da área utilizada da memória - você pode perder um disquete), através do comando "BSAVE "CAS:..."", para ser recuperado mais tarde através do comando "BLOAD "CAS:"". A sub-rotina **ADOTE** da rotina principal deve ser gravada junto com o conjunto de caracteres, para que, quando este for carregado de volta, o sistema o adote.

A rotina:

```
ENDER.   CÓDIGOS MNEMÔNICA          COMENTÁRIOS
(LABEL)
EDITCAR.
E000     CDF6E0  CALL INICIA        Partida quente
CAR1.
E003     CDBDE0  CALL AMPLIA        Amplia caractere
E006     CDFEE1  CALL COORDS        Coordenadas
E009     1608    LD D,8             Tamanho cursor
E00B     CD2FE2  CALL TECLA         Chama comandos
E00E     FE51    CP "Q"             Se for tecla
E010     C8      RET Z              "Q" retorna
E011     2103E0  LD HL, CH1         Endereço inicial
E014     E5      PUSH HL            Guarda HL
E015     FE41    CP "A"             Se for tecla A
E017     CA6EE2  JP Z, ADOTE        Adota conjunto
E01A     FE0D    CP CR              Se for RETURN
E01C     281F    JR Z, EDICÃO       Chama EDICÃO
E01E     0E01    LD C,1             C=offset
E020     FE1C    CP DIREITA         Se for cursor a
E022     2811    JR Z, CH2          dir. salta CH2
E024     0EFF    LD C, FF           Se for cursor a
E026     FE1D    CP ESQUERDA        esq. salta CH2
E028     280B    JR Z, CH2
E02A     0EF0    LD C, F0
E02C     FE1E    CP CIMA            Se for cursor
E02E     2805    JR Z, CH2          p/ cima salta
E030     0E10    LD C, 16           p/ CH2
```

```
E032    FE1F     CP BAIXO        Se for cursor
E034    C0       RET NZ          p/ baixo return
CAR2.
E035    3AA1E2   LD A,(NCAR)     Caractere atual
E038    81       ADD A,C         Soma offset
E039    32A1E2   LD (NCAR),A     Novo caractere
E03C    C9       RET

EDICÃO
E03D    CDE6E1   CALL PIXY       Coordenadas
E040    1602     LD D,2          Tam. cursor
E042    CD2FE2   CALL TECLA      Chama comandos
E045    FE0D     CP RETURN       Retorna se for
E047    C8       RET Z           RETURN
E048    213DE0   LD HL, EDIÇ     HL=E03D
E04B    E5       PUSH HL         Salva HL
E04C    0100FE   LD BC,FE00      Máscaras AND/OR
E04F    FE20     CP SPACE
E051    2824     JR Z, EDIT3
E053    0C       INC C           Másc. OR nova
E054    FE2E     CP "."          Ponto
E056    281F     JR Z, EDIT3
E058    FE1C     CP DIREITA      Cursor à direita
E05A    2811     JR Z, EDIT2
E05C    0EFF     LD C,FF         C=FF
E05E    FE1D     CP ESQUERDA     Cursor à esquerda
E060    280B     JR Z, EDIT2
E062    0EF8     LD C,F8
E064    FE1E     CP CIMA         Cursor p/ cima
E066    2805     JR Z, EDIT2
E068    0E08     LD C,8
E06A    FE1F     CP BAIXO        Cursor p/ baixo
E06C    C0       RET NZ

EDIT2.
E06D    3AA2E2   LD A,(NPIX)     Pixel atual
E070    81       ADD A,C         Soma C
E071    E63F     AND 63          Envolve entorno
E073    32A2E2   LD (NPIX),A     Novo pixel
E076    C9       RET
```

```
EDIT3.
E077    CD1EE2  CALL PADRÃO     Padrão em IY
E07A    3AA2E2  LD A,(NPIX)     Pixel atual
E07D    F5      PUSH AF         Salva AF
E07E    0F      RRCA
E07F    0F      RRCA
E080    0F      RRCA
E081    E607    AND 7           A=linha
E083    5F      LD E,A
E084    1600    LD D,0          DE=linha
E086    FD19    ADD IY,DE       Posição
E088    F1      POP AF
E089    E607    AND 7           A=coluna
E08B    3C      INC A

EDIT4.
E08C    CB08    RRC B           Máscara AND
E08E    CB09    RRC C           Máscara OR
E090    3D      DEC A           Conta colunas
E091    20F9    JR NZ,EDIT4
E093    FD7E00  LD A,(IY+0)     A=Padrão
E096    A0      AND B           Compara bit
E097    B1      OR C            Novo bit
E098    FD7700  LD (IY+0),A     Novo padrão
E09B    CDBDE0  CALL AMPLIA     Atualiza AMPLIA

CAROUT.
E09E    CD1EE2  CALL PADRÃO     IY=Padrão
E0A1    CDFEE1  CALL COORDS     Pega coorden.
E0A4    CDA3E1  CALL MAP
E0A7    0608    LD B,8          Num. lin. pix.

COLUNA
E0A9    D5      PUSH DE         Salva DE
E0AA    E5      PUSH HL         Salva HL
E0AB    3E08    LD A,8          Num. col. pix.
```

```
E0AD    FD5E00  LD E,(IY+0)      E=Padrão
E0B0    CDC4E1  CALL SETLIN      Ajusta linha
E0B3    E1      POP HL           HL=CLOC
E0B4    D1      POP DE           D=CMASK
E0B5    CDB8E1  CALL DESCE       Desce pixel
E0B8    FD23    INC IY
E0BA    10ED    DJNZ COLUNA
E0BC    C9      RET

AMPLIA
E0BD    CD1EE2  CALL PADRÃO      IY=Padrão
E0C0    0EBF    LD C,191         Inicia X
E0C2    1E07    LD E,7           Inicia Y
E0C4    CDA3E1  CALL MAP
E0C7    0608    LD B,8           Num. lin. pix.

CM1:
E0C9    0E05    ld c,5           Ampliar linha

CM2:
E0CB    C5      PUSH BC
E0CC    D5      PUSH DE
E0CD    E5      PUSH HL
E0CE    0608    LD B,8
E0D0    FD7E00  LD A,(IY+0)      A=Padrão

CM3:
E0D3    07      RLCA             Testa bit
E0D4    F5      PUSH AF
E0D5    9F      SBC A,A          0=0, 1=FF
E0D6    5F      LD E,A           E=ampliação
E0D7    3E05    LD A,5           Ampliação col.
E0D9    CDC4E1  CALL SETLIN      Ajusta linha
E0DC    CDAEE1  CALL DIREIT      Pixel a dir.
E0DF    CDAEE1  CALL DIREIT
E0E2    F1      POP AF
E0E3    10EE    DJNZ CM3
E0E5    E1      POP HL
E0E6    D1      POP DE
```

```
E0E7    C1      POP BC
E0E8    CDB8E1  CALL DESCE          Desce pixel
E0EB    0D      DEC C
E0EC    20DD    JR NZ, CM2
E0EE    CDB8E1  CALL DESCE
E0F1    FD23    INC IY
E0F3    10D4    DJNZ CM1
E0F5    C9      RET

INICIA
E0F6    010008  LD BC,2048          Tamanho
E0F9    11A3E2  LD DE,CHRTA         Destino
E0FC    2A20F9  LD HL,(CGPNT+1)     Fonte
IN1:
E0FF    C5      PUSH BC
E100    D5      PUSH DE
E101    3A1FF9  LD A,(CGPNT)
E104    CD0C00  CALL RDSLT          Lê padrão car.
E107    FB      EI
E108    D1      POP DE
E109    C1      POP BC
E10A    12      LD (DE),A           Põe no buffer
E10B    13      INC DE
E10C    23      INC HL
E10D    0B      DEC BC
E10E    78      LD A,B
E10F    B1      OR C
E110    20ED    JR NZ, IN1
E112    CD7200  CALL INIGRP         SCREEN 2
E115    3E0100  LD A,01             Cor da frente 1
E118    07      RLCA
E119    07      RLCA
E11A    07      RLCA
E11B    07      RLCA
E11C    4F      LD C,A              C=Cor 1
E11D    3E0F00  LD A,0F             Fundo=cor 15
E120    B1      OR C
E121    010018  LD BC,6144          Tam tab. cor
E124    2AC9F3  LD HL,(GRPCOL)      Tabela cores
E127    CD5600  CALL FILVRM         Preenche cores
E12A    210BB1  LD HL,177*256+11
```

```
E12D    Ø10AFF  LD BC,FF*256+190
E130    1E06    LD E,6
E132    3E11    LD A,17
E134    CD62E1  CALL GRELHA        Desenha grelha
E137    210631  LD HL,49*256+6
E13A    Ø1BEAA  LD BC,AA*256+190
D13D    1E06    LD E,6
E13F    3E09    LD A,9
E141    CD62E1  CALL GRELHA        Des. grel. ampl
E144    213031  LD HL,49*256+48
E147    Ø1BEFF  LD BC,FF*256+190
E14A    1E06    LD E,6
E14C    3E02    LD A,2
E14E    CD62E1  CALL GRELHA        Desenha grelha
E151    AF      XOR A
E152    32A2E2  LD(NPIX),A         Pixel atual
E155    21A1E2  LD HL, NUMCAR
E158    77      LD (HL),A          Caractere atual

IN2:
E159    E5      PUSH HL
E15A    CD9EEØ  CALL CAROUT        Mostra caractere
E15D    E1      POP HL
E15E    34      INC (HL)           Próximo carac.
E15F    20F8    JR NZ, IN2         Repete 256 x
E161    C9      RET

GRELHA
E162    F5      PUSH AF
E163    C5      PUSH BC
E164    E5      PUSH HL
E165    CDA3E1  CALL MAP
E168    C1      POP BC             B=Comp;C=Passo
E169    F1      POP AF
E16A    5F      LD E,A             E=Padrão
E16B    F1      POP AF
E16C    F5      PUSH AF
E16D    D5      PUSH DE
E16E    E5      PUSH HL

GR1:
```

```
E16F   F5       PUSH AF
E170   C5       PUSH BC
E171   D5       PUSH DE
E172   E5       PUSH HL
E173   78       LD A,B           A=Comprimento
E174   CDC4E1   CALL SETLIN      Ajusta linha
E177   E1       POP HL
E178   D1       POP DE

GR3:
E179   CDB8E1   CALL DESCE       Desce pixel
E17C   0D       DEC C            Executa passo?
E17D   20FA     JR NZ, GR3
E17F   C1       POP BC
E180   F1       POP AF           A=Contador
E181   3D       DEC A            Executa linhas?
E182   20EB     JR NZ,GR1
E184   E1       POP HL           HL=CLOC inicial
E185   D1       POP DE           DE=CMASK inic.
E186   F1       POP AF           A=contador

GR4:
E187   F5       PUSH AF
E188   C5       PUSH BC
E189   D5       PUSH DE
E18A   E5       PUSH HL

GR5:
E18B   3E01     LD A,1           Largura da lin.
E18D   CDC4E1   CALL SETLIN      Linha fina
E190   CDB8E1   CALL DESCE       Desce pixel
E193   10F6     DJNZ GR5         Comprim. vert.
E195   E1       POP HL
E196   D1       POP DE

GR6:
E197   CDAEE1   CALL DIREIT      Pixel à dir.
E19A   0D       DEC C            Executa passo?
E19B   20FA     JR NZ,GR6
E19D   C1       POP BC
E19E   F1       POP AF           A=Contador
```

```
E19F    3D        DEC A          Executa linhas?
E1A0    20E5      JR NZ,GR4
E1A2    C9        RET

MAP:
E1A3    0600      LD B,0         X byte + sign.
E1A5    50        LD D,B         Y byte - sign.
E1A6    CD1101    CALL MAPXYC    Coordenadas
E1A9    CD1401    CALL FETCHC    HL=CLOC
E1AC    57        LD D,A         D=CMASK
E1AD    C9        RET

DIREIT:
E1AE    CB0A      RRC D          Shift CMASK
E1B0    D0        RET NC         NC=mesma célula

RP1:
E1B1    C5        PUSH BC
E1B2    010800    LD BC,8
E1B5    09        ADD HL,BC      HL=próx. célula
E1B6    C1        POP BC
E1B7    C9        RET

DESCE:
E1B8    23        INC HL         Increm. CLOC
E1B9    7D        LD A,L
E1BA    E607      AND 7          Selec. lin. pix
E1BC    C0        RET NZ         NZ=mesma célula
E1BD    C5        PUSH BC
E1BE    01F800    LD BC,F8
E1C1    09        ADD HL,BC      HL=próx. célula
E1C2    C1        POP BC
E1C3    C9        RET

SETLIN:
E1C4    C5        PUSH BC
E1C5    47        LD B,A         B=contador

SE1:
E1C6    CD4A00    CALL RDVRM     Padrão antigo
```

```
SE2:
E1C9    4F       LD C,A          C=antigo
E1CA    7A       LD A,D          A=CMASK
E1CB    2F       CPL             Máscara AND
E1CC    A1       AND C           Bit antigo
E1CD    CB03     RLC E           Shift padrão
E1CF    3001     JR NC,SE3       NC=pixel 0
E1D1    B2       OR D            Seta pixel 1

SE3:
E1D2    05       DEC B           Terminou?
E1D3    280C     JR Z,SE4
E1D5    CB0A     RRC D           CMASK à direita
E1D7    30F0     JR NC,SE2       NC=mesma célula
E1D9    CD4D00   CALL WRTVRM     Atualiza célula
E1DC    CDB1E1   CALL RP1        Próxima célula
E1DF    18E5     JR SE1          Recomeça

SE4:
E1E1    CD4D00   CALL WRTVRM     Atualiza célula
E1E4    C1       POP BC
E1E5    C9       RET

PIXY:
E1E6    3AA2E2   LD A,(NPIX)     Pixel atual
E1E9    F5       PUSH AF
E1EA    E607     AND 7           Coluna
E1EC    07       RLCA
E1ED    4F       LD C,A          C=Col*2
E1EE    07       RLCA            A=Col*4
E1EF    81       ADD A,C         A=Col*6
E1F0    C6BF     ADD A,191       Início grelha
E1F2    4F       LD C,A          C=Coorden. X
E1F3    F1       POP AF
E1F4    E638     AND 38          Linha*8
E1F6    0F       RRCA
E1F7    5F       LD E,A          E=Linha*4
```

```
E1F8    ØF       RRCA              A=Linha*2
E1F9    83       ADD A,E           A=Linha*6
E1FA    C6Ø7     ADD A,7           Inicia grelha
E1FC    5F       LD E,A            E=Coorden. Y
E1FD    C9       RET

COORDS:
E1FE    3AA1E2   LD A,(NCAR)       Caractere atual
E2Ø1    F5       PUSH AF
E2Ø2    CD14E2   CALL MULTI1       Coluna*11
E2Ø5    C6ØC     ADD A,12          Inicia grelha
E2Ø7    4F       LD C,A            C=Coorden. X
E2Ø8    F1       POP AF
E2Ø9    ØF       RRCA
E2ØA    ØF       RRCA
E2ØB    ØF       RRCA
E2ØC    ØF       RRCA
E2ØD    CD14E2   CALL MULTI1       Linha*11
E21Ø    C6Ø8     ADD A,8           Inicia grelha
E212    5F       LD E,A            E=Coorden. Y
E213    C9       RET

MULTI1:
E214    E6ØF     AND OF
E216    57       LD D,A            D=N
E217    Ø7       RLCA
E218    47       LD B,A            B=N*2
E219    Ø7       RLCA
E21A    Ø7       RLCA              A=N*8
E21B    8Ø       ADD A,B
E21C    82       ADD A,D           A=N*11
E21D    C9       RET

PADRÃO:
E21E    3AA1E2   LD A,(NCAR)       Caractere atual
E221    6F       LD L,A
E222    26ØØ     LD H,Ø            HL=caractere
E224    29       ADD HL,HL
E225    29       ADD HL,HL
E226    29       ADD HL,HL         HL=car*8
E227    EB       EX DE,HL          DE=car*8
```

```
E228    FD21A3E2 LD IY, CARTAB  Padrões
E22C    FD19     ADD IY,DE      Padrão em IY
E22E    C9       RET

TECLA:
E22F    0600     LD B,0         Flag do cursor

GE1:
E231    C5       PUSH BC        C=Coorden. X
E232    D5       PUSH DE        E=Coorden. Y
E233    CD50E2   CALL INVERT    Pisca cursor
E236    D1       POP DE
E237    C1       POP BC
E238    04       INC B
E239    21401F   LD HL,8000

GE2:
E23C    CD9C00   CALL CHSNS     Checa KEYBUF
E23F    2007     JR NZ,GE3
E241    2B       DEC HL
E242    7C       LD A,H
E243    B5       OR L
E244    20F6     JR NZ,GE2
E246    18E9     JR GE1         Tempo p/ cursor

GE3:
E248    CB40     BIT 0,B        Est. do cursor
E24A    C450E2   CALL NZ,INVERT Remove curs.
E24D    C39F00   JP CHGET       Coleta caractere

INVERT:
E250    D5       PUSH DE
E251    CDA3E1   CALL MAP       Coordenadas
E254    F1       POP AF         A=Tamanho curs.
E255    47       LD B,A         B=Linhas
E256    5F       LD E,A         E=Colunas

IV1:
E257    D5       PUSH DE
E258    E5       PUSH HL
```

```
IV2:
E259    CD4A00  CALL RDVRM      Lê padrão ant.
E25C    AA      XOR D           Verifica bit
E25D    CD4D00  CALL WRTVRM     Escreve de novo
E260    CDAEE1  CALL DIREIT     Pixel à direita
E263    1D      DEC E
E264    20F3    JR NZ,IV2
E266    E1      POP HL          HL=CLOC
E267    D1      POP DE          D=CMASK
E268    CDB8E1  CALL DESCE      Desce 1 pixel
E26B    10EA    DJNZ IV1
E26D    C9      RET

ADOTE:
E26E    010008  LD BC,2048      Tamanho
E271    1180EB  LD DE,EB80      Destino
E274    ED5320F9LD (CGPNT+1),DE
E278    21A3E2  LD HL,CHRTAB    Fonte
E27B    EDB0    LDIR            Copia bloco
E27D    CD3801  CALL RLSREG     Lê reg. PSLOT
E280    07      RLCA
E281    07      RLCA
E282    E603    AND 3           Selec. página3
E284    4F      LD C,A
E285    0600    LD B,0          BC=página3PSLOT
E287    21C1FC  LD HL,EXPTBL    Expansores
E28A    09      ADD HL,BC
E28B    CB7E    BIT 7,(HL)      PSLOT Expandido
E28D    280E    JR Z, AD1       Z=normal
E28F    21C5FC  LD HL,SLTTBL    Reg. secundár.
E292    09      ADD HL,BC
E293    7E      LD A,(HL)       A=reg.secund.
E294    07      RLCA
E295    07      RLCA
E296    07      RLCA
E297    07      RLCA
E298    E60C    AND 0C          A=página3SSLOT
E29A    B1      OR C
E29B    CBFF    SET 7,A

AD1:
```

```
E29D     321FF9  LD (CGPNT),A
E2A0     C9      RET

E2A1     00      NCAR              Caractere atual
E2A2     00      NPIXEL            Pixel atual
E2A3                               Padrões até
                                   EAA2
```

E agora, para encerrar o livro, duas pequenas rotinas que servem para ler/gravar programas armazenados em fita cassete, sem header, que serão úteis se você quiser fazer uma cópia da fita que mais gosta.

Note apenas que estas possuem capacidade de ler apenas 16K, e conseqüentemente só gravam também 16K.

Mas, se você entendeu a teoria deste livro, principalmente no que se refere a slots e paginação da memória, você será perfeitamente capaz de alterar estas rotinas com o propósito de ampliar sua capacidade para 32K ou mais. Não é difícil.

Vale a pena tentar.

Repare também que nas rotinas o endereço inicial é sempre &H9000 (armazenado no par HL) e o comprimento ou endereço final é sempre &H3FFF (armazenado em DE). Eles podem perfeitamente ser alterados.

A título de exemplo, as rotinas podem ser armazenadas em uma área antes do endereço &H9000. Por isso, não coloquei nenhum endereço de armazenamento, já que elas utilizam endereçamento relativo e portanto podem ser locadas em qualquer área da memória (menos a que será ocupada pelo programa).

ROTINA DE LER CASSETE:

```
F3      DI              Desabilita interrupções

CDE100  CALL TAPION     Call tape input on
210090  LD HL,9000      Endereço    inicial
                        em HL
11FF3F  LD DE,3FFF      Comprimento em DE

CONFERE:
E5      PUSH HL         Salva HL
D5      PUSH DE         Salva DE
CDE400  CALL TAPIN      Call Tape input
D1      POP DE          Recupera DE
E1      POP HL          Recupera HL
77      LD (HL),A       Carrega posição apontada
                        por HL  com conteúdo de A

23      INC HL          Nova posição de HL
1B      DEC DE          Nova posição de DE
7A      LD A,D          Carrega  Acumulador
                        com  conteúdo de D
B3      OR E            Oper. lóg. OR com E
20F2    JR NZ, CONFERE  Se não for Ø, volta
CDE700  CALL TAPIOF     Call tape input off
C9      RET             Retorna
```

ROTINA DE GRAVAR EM CASSETE:

```
F3      DI              Desabilita interrupções
3EFF    LD A,FF         Carrega reg.A c/ FF
CDEA00  CALL TAPOON     Call Tape output on
210090  LD HL,9000      HL armazenando endereço
                        inicial
11FF3F  LD DE,3FFF      DE armazenando comprimento

CONFERE:
7E      LD A,(HL)       Carrega reg. A  com
                        conteúdo da posição
                        apontada por HL
E5      PUSH HL         Salva HL
```

```
D5      PUSH DE          Salva DE
CDEDØØ  CALL TAPOUT      Call Tape output
D1      POP DE           Recupera DE
E1      POP HL           Recupera HL
23      INC HL           Novo  endereço
1B      DEC DE           Novo  comprimento
7B      LD A,E           Carrega A com E
B3      OR E             É igual ?
2ØF2    JR NZ,CONFERE    Se não for, volta
CDFØØØ  CALL TAPOOF      Call tape  output
                         off
C9      RET              Retorna
```

E assim nós encerramos este livro, que espero sinceramente seja de grande valia para você, tanto quanto foi para mim.

Gostaria muito que meus objetivos, ao escrevê-lo, fossem, não digo plenamente, mas satisfatoriamente atingidos, ou seja, que você, ao chegar ao final, entendesse de linguagem de máquina, e conhecesse todos os segredos do seu MSX e de seu sistema eficiente.

De forma nenhuma espero que você escreva grandes programas, jogos, ou mesmo aplicativos em linguagem de máquina, de 1Ø, 12 ou 16K de memória. Seria querer demais! Basta entender e criar programas híbridos, ou seja, em Basic e em Assembler.

Até breve!

APÊNDICE A

CONVERSÃO DE VALORES HEXADECIMAIS, DECIMAIS E BINÁRIOS

HEXADECIMAL	DECIMAL	BINÁRIO
ØØ	Ø	ØØØØØØØØ
Ø1	1	ØØØØØØØ1
Ø2	2	ØØØØØØ1Ø
Ø3	3	ØØØØØØ11
Ø4	4	ØØØØØ1ØØ
Ø5	5	ØØØØØ1Ø1
Ø6	6	ØØØØØ11Ø
Ø7	7	ØØØØØ111
Ø8	8	ØØØØ1ØØØ
Ø9	9	ØØØØ1ØØ1
ØA	1Ø	ØØØØ1Ø1Ø
ØB	11	ØØØØ1Ø11
ØC	12	ØØØØ11ØØ
ØD	13	ØØØØ11Ø1
ØE	14	ØØØØ111Ø
ØF	15	ØØØØ1111
1Ø	16	ØØØ1ØØØØ
11	17	ØØØ1ØØØ1
12	18	ØØØ1ØØ1Ø
13	19	ØØØ1ØØ11
14	2Ø	ØØØ1Ø1ØØ
15	21	ØØØ1Ø1Ø1
16	22	ØØØ1Ø11Ø
17	23	ØØØ1Ø111
18	24	ØØØ11ØØØ
19	25	ØØØ11ØØ1
1A	26	ØØØ11Ø1Ø

1B	27	ØØØ11Ø11
1C	28	ØØØ111ØØ
1D	29	ØØØ111Ø1
1E	3Ø	ØØØ1111Ø
1F	31	ØØØ11111
2Ø	32	ØØ1ØØØØØ
21	33	ØØ1ØØØØ1
22	34	ØØ1ØØØ1Ø
23	35	ØØ1ØØØ11
24	36	ØØ1ØØ1ØØ
25	37	ØØ1ØØ1Ø1
26	38	ØØ1ØØ11Ø
27	39	ØØ1ØØ111
28	4Ø	ØØ1Ø1ØØØ
29	41	ØØ1Ø1ØØ1
2A	42	ØØ1Ø1Ø1Ø
2B	43	ØØ1Ø1Ø11
2C	44	ØØ1Ø11ØØ
2D	45	ØØ1Ø11Ø1
2E	46	ØØ1Ø111Ø
2F	47	ØØ1Ø1111
3Ø	48	ØØ11ØØØØ
31	49	ØØ11ØØØ1
32	5Ø	ØØ11ØØ1Ø
33	51	ØØ11ØØ11
34	52	ØØ11Ø1ØØ
35	53	ØØ11Ø1Ø1
36	54	ØØ11Ø11Ø
37	55	ØØ11Ø111
38	56	ØØ111ØØØ
39	57	ØØ111ØØ1
3A	58	ØØ111Ø1Ø
3B	59	ØØ111Ø11
3C	6Ø	ØØ1111ØØ
3D	61	ØØ1111Ø1
3E	62	ØØ11111Ø
3F	63	ØØ111111
4Ø	64	Ø1ØØØØØØ
41	65	Ø1ØØØØØ1
42	66	Ø1ØØØØ1Ø
43	67	Ø1ØØØØ11

44	68	01000100
45	69	01000101
46	70	01000110
47	71	01000111
48	72	01001000
49	73	01001001
4A	74	01001010
4B	75	01001011
4C	76	01001100
4D	77	01001101
4E	78	01001110
4F	79	01001111
50	80	01010000
51	81	01010001
52	82	01010010
53	83	01010011
54	84	01010100
55	85	01010101
56	86	01010110
57	87	01010111
58	88	01011000
59	89	01011001
5A	90	01011010
5B	91	01011011
5C	92	01011100
5D	93	01011101
5E	94	01011110
5F	95	01011111
60	96	01100000
61	97	01100001
62	98	01100010
63	99	01100011
64	100	01100100
65	101	01100101
66	102	01100110
67	103	01100111
68	104	01101000
69	105	01101001
6A	106	01101010
6B	107	01101011
6C	108	01101100

6D	1Ø9	Ø11Ø11Ø1
6E	11Ø	Ø11Ø111Ø
6F	111	Ø11Ø1111
7Ø	112	Ø111ØØØØ
71	113	Ø111ØØØ1
72	114	Ø111ØØ1Ø
73	115	Ø111ØØ11
74	116	Ø111Ø1ØØ
75	117	Ø111Ø1Ø1
76	118	Ø111Ø11Ø
77	119	Ø111Ø111
78	12Ø	Ø1111ØØØ
79	121	Ø1111ØØ1
7A	122	Ø1111Ø1Ø
7B	123	Ø1111Ø11
7C	124	Ø11111ØØ
7D	125	Ø11111Ø1
7E	126	Ø111111Ø
7F	127	Ø1111111
8Ø	128	1ØØØØØØØ
81	129	1ØØØØØØ1
82	13Ø	1ØØØØØ1Ø
83	131	1ØØØØØ11
84	132	1ØØØØ1ØØ
85	133	1ØØØØ1Ø1
86	134	1ØØØØ11Ø
87	135	1ØØØØ111
88	136	1ØØØ1ØØØ
89	137	1ØØØ1ØØ1
8A	138	1ØØØ1Ø1Ø
8B	139	1ØØØ1Ø11
8C	14Ø	1ØØØ11ØØ
8D	141	1ØØØ11Ø1
8E	142	1ØØØ111Ø
8F	143	1ØØØ1111
9Ø	144	1ØØ1ØØØØ
91	145	1ØØ1ØØØ1
92	146	1ØØ1ØØ1Ø
93	147	1ØØ1ØØ11
94	148	1ØØ1Ø1ØØ
95	149	1ØØ1Ø1Ø1

96	150	10010110
97	151	10010111
98	152	10011000
99	153	10011001
9A	154	10011010
9B	155	10011011
9C	156	10011100
9D	157	10011101
9E	158	10011110
9F	159	10011111
A0	160	10100000
A1	161	10100001
A2	162	10100010
A3	163	10100011
A4	164	10100100
A5	165	10100101
A6	166	10100110
A7	167	10100111
A8	168	10101000
A9	169	10101000
AA	170	10101010
AB	171	10101011
AC	172	10101100
AD	173	10101101
AE	174	10101110
AF	175	10101111
B0	176	10110000
B1	177	10110001
B2	178	10110010
B3	179	10110011
B4	180	10110100
B5	181	10110101
B6	182	10110110
B7	183	10110100
B8	184	10111000
B9	185	10111001
BA	186	10111010
BB	187	10111011
BC	188	10111100
BD	189	10111101
BE	190	10111110

BF	191	10111111
C0	192	11000000
C1	193	11000001
C2	194	11000010
C3	195	11000011
C4	196	11000100
C5	197	11000101
C6	198	11000110
C7	199	11000111
C8	200	11001000
C9	201	11001001
CA	202	11001010
CB	203	11001011
CC	204	11001100
CD	205	11001101
CE	206	11001110
CF	207	11001111
D0	208	11010000
D1	209	11010001
D2	210	11010010
D3	211	11010011
D4	212	11010100
D5	213	11010101
D6	214	11010110
D7	215	11010111
D8	216	11011000
D9	217	11011001
DA	218	11011010
DB	219	11011011
DC	220	11011100
DD	221	11011101
DE	222	11011110
DF	223	11011111
E0	224	11100000
E1	225	11100001
E2	226	11100010
E3	227	11100011
E4	228	11100100
E5	229	11100101
E6	230	11100110
E7	231	11100111

E8	232	111Ø1ØØØ
E9	233	111Ø1ØØ1
EA	234	111Ø1Ø1Ø
EB	235	111Ø1Ø11
EC	236	111Ø11ØØ
ED	237	111Ø11Ø1
EE	238	111Ø111Ø
EF	239	111Ø1111
FØ	24Ø	1111ØØØØ
F1	241	1111ØØØ1
F2	242	1111ØØ1Ø
F3	243	1111ØØ11
F4	244	1111Ø1ØØ
F5	245	1111Ø1Ø1
F6	246	1111Ø11Ø
F7	247	1111Ø111
F8	248	11111ØØØ
F9	249	11111ØØ1
FA	25Ø	11111Ø1Ø
FB	251	11111Ø11
FC	252	111111ØØ
FD	253	111111Ø1
FE	254	1111111Ø
FF	255	11111111

APÊNDICE B

CÓDIGOS DE OPERAÇÃO DO Z8Ø ORDENADOS POR MNEMÔNICAS

NOTAS:

D -deslocamento na faixa de -127 a +128
XX -valor equivalente a 1 byte, na faixa de Ø a 255
XXXX-valor equivalente a 2 bytes, na faixa de Ø a 65535

CÓDIGO DE OPERAÇÃO	CÓDIGO HEXADECIMAL	CÓDIGO DECIMAL
ADC A,(HL)	8E	142
ADC A,(IX+D)	DD8E D	221 142 D
ADC A,(IY+D)	FD8E D	253 142 D
ADC A,A	8F	143
ADC A,B	88	136
ADC A,C	89	137
ADC A,D	8A	138
ADC A,E	8B	139
ADC A,H	8C	14Ø
ADC A,L	8D	141
ADC A,XX	CE XX	2Ø6 XX
ADC HL,BC	ED4A	237 74
ADC HL,DE	ED5A	237 9Ø
ADC HL,HL	ED6A	237 1Ø6
ADC HL,SP	ED7A	237 122
ADD A,(HL)	86	134
ADD A,(IX+D)	DD86 D	221 134 D
ADD A,(IY+D)	FD86 D	235 134 D
ADD A,A	87	135
ADD A,B	8Ø	128

ADD A,C	81	129
ADD A,D	82	13Ø
ADD A,E	83	131
ADD A,H	84	132
ADD A,L	85	133
ADD A,XX	C6 XX	198 XX
ADD HL,BC	Ø9	9
ADD HL,DE	19	25
ADD HL,HL	29	41
ADD HL,SP	39	57
ADD IX,BC	DDØ9	221 9
ADD IX,DE	DD19	221 25
ADD IX,HL	DD29	221 41
ADD IX,SP	DD39	221 57
ADD IY,BC	FDØ9	253 9
ADD IY,DE	FD19	253 25
ADD IY,HL	FD29	253 41
ADD IY,SP	FD39	253 57
AND (HL)	A6	166
AND (IX+D)	DDA6 D	221 166 D
AND (IY+D)	FDA6 D	253 166 D
AND A	A7	167
AND B	AØ	16Ø
AND C	A1	161
AND D	A2	162
AND E	A3	163
AND H	A4	164
AND L	A5	165
AND XX	E6 XX	23Ø XX
BIT Ø, (HL)	CB46	2Ø3 7Ø
BIT Ø, (IX+D)	DDCB D 46	221 2Ø3 D 7Ø
BIT Ø, (IY+D)	FDCB D 46	253 2Ø3 D 7Ø
BIT Ø, A	CB47	2Ø3 71
BIT Ø, B	CB4Ø	2Ø3 64
BIT Ø, C	CB41	2Ø3 65
BIT Ø, D	CB42	2Ø3 66
BIT Ø, E	CB43	2Ø3 67
BIT Ø, H	CB44	2Ø3 68
BIT Ø, L	CB45	2Ø3 69

```
BIT 1, (HL)        CB4E           203 70
BIT 1, (IX+D)      DDCB D 4E      221 203 D 70
BIT 1, (IY+D)      FDCB D 4E      253 203 D 70
BIT 1, A           CB4F           203 79
BIT 1, B           CB48           203 72
BIT 1, C           CB49           203 73
BIT 1, D           CB4A           203 74
BIT 1, E           CB4B           203 75
BIT 1, H           CB4C           203 76
BIT 1, L           CB4D           203 77

BIT 2, (HL)        CB56           203 86
BIT 2, (IX+D)      DDCB D 56      221 203 D 86
BIT 2, (IY+D)      FDCB D 56      253 203 D 86
BIT 2, A           CB57           203 87
BIT 2, B           CB50           203 80
BIT 2, C           CB51           203 81
BIT 2, D           CB52           203 82
BIT 2, E           CB53           203 83
BIT 2, H           CB54           203 84
BIT 2, L           CB55           203 85

BIT 3, (HL)        CB5E           203 94
BIT 3, (IX+D)      DDCB D 5E      221 203 D 94
BIT 3, (IY+D)      FDCB D 5E      253 203 D 94
BIT 3, A           CB5F           203 95
BIT 3, B           CB58           203 88
BIT 3, C           CB59           203 89
BIT 3, D           CB5A           203 90
BIT 3, E           CB5B           203 91
BIT 3, H           CB5C           203 92
BIT 3, L           CB5D           203 93

BIT 4, (HL)        CB66           203 102
BIT 4, (IX+D)      DDCB D 66      221 203 D 102
BIT 4, (IY+D)      FDCB D 66      253 203 D 102
BIT 4, A           CB67           203 103
BIT 4, B           CB60           203 96
BIT 4, C           CB61           203 97
BIT 4, D           CB62           203 98
```

BIT 4, E	CB63	203 99
BIT 4, H	CB64	203 100
BIT 4, L	CB65	203 101
BIT 5, (HL)	CB6E	203 110
BIT 5, (IX+D)	DDCB D 6E	221 203 D 110
BIT 5, (IY+D)	FDCB D 6E	253 203 D 110
BIT 5, A	CB6F	203 111
BIT 5, B	CB68	203 104
BIT 5, C	CB69	203 105
BIT 5, D	CB6A	203 106
BIT 5, E	CB6B	203 107
BIT 5, H	CB6C	203 108
BIT 5, L	CB6D	203 109
BIT 6, (HL)	CB76	203 118
BIT 6, (IX+D)	DDCB D 76	221 203 D 118
BIT 6, (IY+D)	FDCB D 76	253 203 D 118
BIT 6, A	CB77	203 119
BIT 6, B	CB70	203 112
BIT 6, C	CB71	203 113
BIT 6, D	CB72	203 114
BIT 6, E	CB73	203 115
BIT 6, H	CB74	203 116
BIT 6, L	CB75	203 117
BIT 7, (HL)	CB7E	203 116
BIT 7, (IX+D)	DDCB D 7E	221 203 D 116
BIT 7, (IY+D)	FDCB D 7E	253 203 D 116
BIT 7, A	CB7F	203 127
BIT 7, B	CB78	203 120
BIT 7, C	CB79	203 121
BIT 7, D	CB7A	203 122
BIT 7, E	CB7B	203 123
BIT 7, H	CB7C	203 124
BIT 7, L	CB7D	203 125
CALL XXXX	CD XXXX	205 XXXX
CALL C, XXXX	DC XXXX	220 XXXX
CALL M, XXXX	FC XXXX	252 XXXX
CALL NC, XXXX	D4 XXXX	212 XXXX
CALL NZ, XXXX	C4 XXXX	196 XXXX

CALL P, XXXX	F4 XXXX	244 XXXX
CALL PE, XXXX	EC XXXX	EC XXXX
CALL PO, XXXX	E4 XXXX	E4 XXXX
CALL Z, XXXX	CC XXXX	CC XXXX
CP (HL)	BE	19Ø
CP (IX+D)	DDBE D	221 19Ø D
CP (IY+D)	FDBE D	253 19Ø D
CP A	BF	191
CP B	B8	184
CP C	B9	185
CP D	BA	186
CP E	BB	187
CP H	BC	188
CP L	BD	189
CP XX	FE XX	254 XX
CPD	EDA9	237 169
CPDR	EDB9	237 185
CPI	EDA1	237 161
CPIR	EDB1	237 177
CPL	2F	47
DAA	27	39
DEC (HL)	35	53
DEC (IX+D)	DD35 D	221 53 D
DEC (IY+D)	FD35 D	253 53 D
DEC A	3D	61
DEC B	Ø5	5
DEC BC	ØB	11
DEC C	ØD	13
DEC D	15	21
DEC DE	1B	27
DEC E	1D	29
DEC H	25	37
DEC HL	2B	43
DEC IX	DD2B	221 43
DEC IY	FD2B	253 43
DEC L	2D	45
DEC SP	3B	59

```
DI              F3          243

DJNZ XX         1Ø XX       16 XX

EI              FB          251

EX (SP),HL      E3          227
EX (SP),IX      DDE3        221 227
EX (SP),IY      FDE3        253 227
EX AF, AF'      Ø8          8
EX DE, HL       EB          235
EXX             D9          217

HALT            76          118

IM Ø            ED46        237 7Ø
IM 1            ED56        237 86
IM 2            ED5E        237 94

IN A, (C)       ED78        237 12Ø
IN A, (XX)      DB XX       219 XX
IN B, (C)       ED4Ø        237 64
IN C, (C)       ED48        237 72
IN D, (C)       ED5Ø        237 8Ø
IN E, (C)       ED58        237 88
IN H, (C)       ED6Ø        237 96
IN L, (C)       ED68        237 1Ø4

INC (HL)        34          52
INC (IX+D)      DD34 D      221 52 D
INC (IY+D)      FD34 D      253 52 D
INC A           3C          6Ø
INC B           Ø4          4
INC BC          Ø3          3
INC C           ØC          12
INC D           14          2Ø
INC DE          13          19
INC E           1C          28
INC H           24          36
INC HL          23          35
INC IX          DD23        221 35
```

```
INC IY          FD23         253 35
INC L           2C           44
INC SP          33           51
IND             EDAA         237 17Ø
INDR            EDBA         237 186
INI             EDA2         237 162
INIR            EDB2         237 178

JP (HL)         E9           233
JP (IX)         DDE9         221 233
JP (IY)         FDE9         253 233
JP C, XXXX      DA XXXX      218 XXXX
JP M, XXXX      FA XXXX      25Ø XXXX
JP NC, XXXX     D2 XXXX      21Ø XXXX
JP NZ, XXXX     C2 XXXX      194 XXXX
JP P, XXXX      F2 XXXX      242 XXXX
JP PE, XXXX     EA XXXX      234 XXXX
JP PO, XXXX     E2 XXXX      226 XXXX
JP XXXX         C3 XXXX      195 XXXX
JP Z, XXXX      CA XXXX      2Ø2 XXXX
JR C, XX        38 XX        56 XX
JR NC, XX       3Ø XX        48 XX
JR NZ, XX       2Ø XX        32 XX
JR XX           18 XX        24 XX
JR Z, XX        28 XX        4Ø XX

LD (BC), A      Ø2           2
LD (DE), A      12           18
LD HL, (XXXX)   2A XXXX      42 XXXX
LD (HL), A      77           119
LD (HL), B      7Ø           112
LD (HL), C      71           113
LD (HL), D      72           114
LD (HL), E      73           115
LD (HL), H      74           116
LD (HL), L      75           117
LD (HL), XX     36 XX        54 XX
LD (IX+D), A    DD77 D       221 119 D
LD (IX+D), B    DD7Ø D       221 112 D
LD (IX+D), C    DD71 D       221 113 D
LD (IX+D), D    DD72 D       221 114 D
```

LD (IX+D), E	DD73 D	221 115 D
LD (IX+D), H	DD74 D	221 116 D
LD (IX+D), L	DD75 D	221 117 D
LD (IX+D), XX	DD36 D XX	221 54 D XX
LD (IY+D), A	FD77 D	253 119 D
LD (IY+D), B	FD7Ø D	253 112 D
LD (IY+D), C	FD71 D	253 113 D
LD (IY+D), D	FD72 D	253 114 D
LD (IY+D), E	FD73 D	253 115 D
LD (IY+D), H	FD74 D	253 116 D
LD (IY+D), L	FD75 D	253 117 D
LD (IY+D), XX	FD36 D XX	253 54 D XX
LD (XXXX), A	32 XXXX	5Ø XXXX
LD (XXXX), BC	ED43 XXXX	237 67 XXXX
LD (XXXX), DE	ED53 XXXX	237 83 XXXX
LD (XXXX), HL	22 XXXX	34 XXXX
LD (XXXX), IX	DD22 XXXX	221 34 XXXX
LD (XXXX), IY	FD22 XXXX	253 34 XXXX
LD (XXXX), SP	ED73 XXXX	237 115 XXXX
LD A, (BC)	ØA	1Ø
LD A, (DE)	1A	26
LD A, (HL)	7E	126
LD A, (IX+D)	DD7E D	221 126 D
LD A, (IY+D)	FD7E D	253 126 D
LD A, (XXXX)	3A XXXX	58 XXXX
LD A, A	7F	127
LD A, B	78	12Ø
LD A, C	79	121
LD A, D	7A	122
LD A, E	7B	123
LD A, H	7C	124
LD A, I	ED57	237 87
LD A, L	7D	125
LD A, R	ED5F	237 85
LD A, XX	3E XX	62 XX
LD B, (HL)	46	7Ø
LD B, (IX+D)	DD46 D	221 7Ø D
LD B, (IY+D)	FD46 D	253 7Ø D
LD B, A	47	71

LD B, B	4Ø	64
LD B, C	41	65
LD B, D	42	66
LD B, E	43	67
LD B, H	44	68
LD B, L	45	69
LD B, XX	Ø6 XX	6 XX
LD BC, (XXXX)	ED4B XXXX	237 75 XXXX
LD BC, XXXX	Ø1 XXXX	1 XXXX
LD C, (HL)	4E	78
LD C, (IX+D)	DD4E D	221 78 D
LD C, (IY+D)	FD4E D	253 78 D
LD C, A	4F	79
LD C, B	48	72
LD C, C	49	73
LD C, D	4A	74
LD C, E	4B	75
LD C, H	4C	76
LD C, L	4D	77
LD C, XX	ØE XX	14 XX
LD D, (HL)	56	86
LD D, (IX+D)	DD56 D	221 86 D
LD D, (IY+D)	FD56 D	253 86 D
LD D, A	57	87
LD D, B	5Ø	8Ø
LD D, C	51	81
LD D, D	52	82
LD D, E	53	83
LD D, H	54	84
LD D, L	55	85
LD D, XX	16 XX	22 XX
LD DE, (XXXX)	ED5B XXXX	237 91 XXXX
LD DE, XXXX	11 XXXX	17 XXXX
LD E, (HL)	5E	94
LD E, (IX+D)	DD5E D	221 94 D
LD E, (IY+D)	FD5E D	253 94 D
LD E, A	5F	95
LD E, B	58	88

LD E, C	59	89
LD E, D	5A	90
LD E, E	5B	91
LD E, H	5C	92
LD E, L	5D	93
LD E, XX	1E XX	30 XX
LD H, (HL)	66	102
LD H, (IX+D)	DD66 D	221 102 D
LD H, (IY+D)	FD66 D	253 102 D
LD H, A	67	103
LD H, B	60	96
LD H, C	61	97
LD H, D	62	98
LD H, E	63	99
LD H, H	64	100
LD H, L	65	101
LD H, XX	26 XX	38 XX
LD HL, XXXX	21 XXXX	33 XXXX
LD HL, (XXXX)	2A XXXX	
LD I, A	ED47	237 71
LD IX, XXXX	DD21 XXXX	221 33 XXXX
LD IX, (XXXX)	DD2A XXXX	221 42 XXXX
LD IY, XXXX	FD21 XXXX	253 33 XXXX
LD IY, (XXXX)	FD2A XXXX	253 42 XXXX
LD L, (HL)	6E	110
LD L, (IX+D)	DD6E D	221 110 D
LD L, (IY+D)	FD6E D	253 110 D
LD L, A	6F	111
LD L, B	68	104
LD L, C	69	105
LD L, D	6A	106
LD L, E	6B	107
LD L, H	6C	108
LD L, L	6D	109
LD L, XX	2E XX	46 XX
LD R, A	ED4F	237 79
LD SP, (XXXX)	ED7B XXXX	237 123 XXXX

LD SP, HL	F9	249
LD SP, IX	DDF9	221 249
LD SP, IY	FDF9	253 249
LD SP, XXXX	31 XXXX	49 XXXX
LDD	EDA8	237 168
LDDR	EDB8	237 184
LDI	EDAØ	237 16Ø
LDIR	EDBØ	237 176
NEG	ED44	237 68
NOP	ØØ	Ø
OR (HL)	B6	182
OR (IX+D)	DDB6 D	221 182 D
OR (IY+D)	FDB6 D	253 182 D
OR A	B7	183
OR B	BØ	176
OR C	B1	177
OR D	B2	178
OR E	B3	179
OR H	B4	18Ø
OR L	B5	181
OR XX	F6 XX	246 XX
OTDR	EDBB	237 187
OTIR	EDB3	237 179
OUT (C), A	ED79	237 121
OUT (C), B	ED41	237 65
OUT (C), C	ED49	237 73
OUT (C), D	ED51	237 81
OUT (C), E	ED59	237 89
OUT (C), H	ED61	237 97
OUT (C), L	ED69	237 1Ø5
OUT (XX), A	D3 XX	211 XX
OUTD	EDAB	237 171
OUTI	EDA3	237 163
POP AF	F1	241

POP BC	C1	193
POP DE	D1	209
POP HL	E1	225
POP IX	DDE1	221 225
POP IY	FDE1	253 225
PUSH AF	F5	245
PUSH BC	C5	197
PUSH DE	D5	213
PUSH HL	E5	229
PUSH IX	DDE5	221 229
PUSH IY	FDE5	253 229
RES Ø, (HL)	CB86	203 134
RES Ø, (IX+D)	DDCB D 86	221 203 D 134
RES Ø, (IY+D)	FDCB D 86	253 203 D 134
RES Ø, A	CB87	203 135
RES Ø, B	CB8Ø	203 128
RES Ø, C	CB81	203 129
RES Ø, D	CB82	203 13Ø
RES Ø, E	CB83	203 131
RES Ø, H	CB84	203 132
RES Ø, L	CB85	203 133
RES 1, (HL)	CB8E	203 142
RES 1, (IX+D)	DDCB D 8E	221 203 D 142
RES 1, (IY+D)	FDCB D 8E	253 203 D 142
RES 1, A	CB8F	203 143
RES 1, B	CB88	203 136
RES 1, C	CB89	203 137
RES 1, D	CB8A	203 138
RES 1, E	CB8B	203 139
RES 1, H	CB8C	203 14Ø
RES 1, L	CB8D	203 141
RES 2, (HL)	CB96	203 15Ø
RES 2, (IX+D)	DDCB D 96	221 203 D 15Ø
RES 2, (IY+D)	FDCB D 96	253 203 D 15Ø
RES 2, A	CB97	203 151

```
RES 2, B          CB9Ø              203 144
RES 2, C          CB91              203 145
RES 2, D          CB92              203 146
RES 2, E          CB93              203 147
RES 2, H          CB94              203 148
RES 2, L          CB95              203 149

RES 3, (HL)       CB9E              203 158
RES 3, (IX+D)     DDCB D 9E         221 2Ø3 D 158
RES 3, (IY+D)     FDCB D 9E         253 2Ø3 D 158
RES 3, A          CB9F              2Ø3 159
RES 3, B          CB98              2Ø3 152
RES 3, C          CB99              2Ø3 153
RES 3, D          CB9A              2Ø3 154
RES 3, E          CB9B              2Ø3 155
RES 3, H          CB9C              2Ø3 156
RES 3, L          CB9D              2Ø3 157

RES 4, (HL)       CBA6              2Ø3 166
RES 4, (IX+D)     DDCB D A6         221 2Ø3 D 166
RES 4, (IY+D)     FDCB D A6         253 2Ø3 D 166
RES 4, A          CBA7              2Ø3 167
RES 4, B          CBAØ              2Ø3 16Ø
RES 4, C          CBA1              2Ø3 161
RES 4, D          CBA2              2Ø3 162
RES 4, E          CBA3              2Ø3 163
RES 4, H          CBA4              2Ø3 164
RES 4, L          CBA5              2Ø3 165

RES 5, (HL)       CBAE              2Ø3 174
RES 5, (IX+D)     DDCB D AE         221 2Ø3 D 174
RES 5, (IY+D)     FDCB D AE         253 2Ø3 D 174
RES 5, A          CBAF              2Ø3 175
RES 5, B          CBA8              2Ø3 168
RES 5, C          CBA9              2Ø3 169
RES 5, D          CBAA              2Ø3 17Ø
RES 5, E          CBAB              2Ø3 171
RES 5, H          CBAC              2Ø3 172
RES 5, L          CBAD              2Ø3 173

RES 6, (HL)       CBB6              2Ø3 182
```

```
RES 6, (IX+D)     DDCB D B6     221 203 D 182
RES 6, (IY+D)     FDCB D B6     253 203 D 182
RES 6, A          CBB7          203 183
RES 6, B          CBB0          203 176
RES 6, C          CBB1          203 177
RES 6, D          CBB2          203 178
RES 6, E          CBB3          203 179
RES 6, H          CBB4          203 180
RES 6, L          CBB5          203 181

RES 7, (HL)       CBBE          203 190
RES 7, (IX+D)     DDCB D BE     221 203 D 190
RES 7, (IY+D)     FDCB D BE     253 203 D 190
RES 7, A          CBBF          203 191
RES 7, B          CBB8          203 184
RES 7, C          CBB9          203 185
RES 7, D          CBBA          203 186
RES 7, E          CBBB          203 187
RES 7, H          CBBC          203 188
RES 7, L          CBBD          203 189

RET               C9            201
RET C             D8            216
RET M             F8            248
RET NC            D0            208
RET NZ            C0            192
RET P             F0            240
RET PE            E8            232
RET PO            E0            224
RET Z             C8            200
RET I             ED4D          237 77
RET N             ED45          237 69

RL (HL)           CB16          203 22
RL (IX+D)         DDCB D 16     221 203 D 22
RL (IY+D)         FDCB D 16     253 203 D 22
RL A              CB17          203 23
RL B              CB10          203 16
RL C              CB11          203 17
RL D              CB12          203 18
RL E              CB13          203 19
```

```
RL H           CB14           203 20
RL L           CB15           203 21
RLA            17             23

RLC (HL)       CBØ6           203 6
RLC (IX+D)     DDCB D Ø6      221 203 D 6
RLC (IY+D)     FDCB D Ø6      253 203 D 6
RLC A          CBØ7           203 7
RLC B          CBØØ           203 Ø
RLC C          CBØ1           203 1
RLC D          CBØ2           203 2
RLC E          CBØ3           203 3
RLC H          CBØ4           203 4
RLC L          CBØ5           203 5
RLCA           Ø7             7
RLD            ED6F           237 111

RR (HL)        CB1E           203 3Ø
RR (IX+D)      DDCB D 1E      221 203 D 3Ø
RR (IY+D)      FDCB D 1E      253 203 D 3Ø
RR A           CB1F           203 31
RR B           CB18           203 24
RR C           CB19           203 25
RR D           CB1A           203 26
RR E           CB1B           203 27
RR H           CB1C           203 28
RR L           CB1D           203 29
RRA            1F             31

RRC (HL)       CBØE           203 14
RRC (IX+D)     DDCB D ØE      221 203 D 14
RRC (IY+D)     FDCB D ØE      253 203 D 14
RRC A          CBØF           203 15
RRC B          CBØ8           203 8
RRC C          CBØ9           203 9
RRC D          CBØA           203 1Ø
RRC E          CBØB           203 11
RRC H          CBØC           203 12
RRC L          CBØD           203 13
RRCA           ØF             15
```

RRD	ED67	237 1Ø3
RST Ø	C7	199
RST 8	CF	2Ø7
RST 1Ø	D7	215
RST 18	DF	223
RST 2Ø	E7	231
RST 28	EF	239
RST 3Ø	F7	247
RST 38	FF	255
SBC A, (HL)	9E	158
SBC A, (IX+D)	DD9E D	221 158 D
SBC A, (IY+D)	FD9E D	253 158 D
SBC A, A	9F	159
SBC A, B	98	152
SBC A, C	99	153
SBC A, D	9A	154
SBC A, E	9B	155
SBC A, H	9C	156
SBC A, L	9D	157
SBC A, XX	DE XX	222 XX
SBC HL, BC	ED42	237 66
SBC HL, DE	ED52	237 82
SBC HL, HL	ED62	237 98
SBC HL, SP	ED72	237 114
SCF	37	55
SET Ø, (HL)	CBC6	2Ø3 198
SET Ø, (IX+D)	DDCB D C6	221 2Ø3 D 198
SET Ø, (IY+D)	FDCB D C6	253 2Ø3 D 198
SET Ø, A	CBC7	2Ø3 199
SET Ø, B	CBCØ	2Ø3 192
SET Ø, C	CBC1	2Ø3 193
SET Ø, D	CBC2	2Ø3 194
SET Ø, E	CBC3	2Ø3 195
SET Ø, H	CBC4	2Ø3 196
SET Ø, L	CBC5	2Ø3 197
SET 1, (HL)	CBCE	2Ø3 2Ø6

```
SET 1, (IX+D)     DDCB D CE      221 203 D 206
SET 1, (IY+D)     FDCB D CE      253 203 D 206
SET 1, A          CBCF           203 207
SET 1, B          CBC8           203 200
SET 1, C          CBC9           203 201
SET 1, D          CBCA           203 202
SET 1, E          CBCB           203 203
SET 1, H          CBCC           203 204
SET 1, L          CBCD           203 205

SET 2, (HL)       CBD6           203 214
SET 2, (IX+D)     DDCB D D6      221 203 D 214
SET 2, (IY+D)     FDCB D D6      253 203 D 214
SET 2, A          CBD7           203 215
SET 2, B          CBD0           203 208
SET 2, C          CBD1           203 209
SET 2, D          CBD2           203 210
SET 2, E          CBD3           203 211
SET 2, H          CBD4           203 212
SET 2, L          CBD5           203 213

SET 3, (HL)       CBDE           203 222
SET 3, (IX+D)     DDCB D DE      221 203 D 222
SET 3, (IY+D)     FDCB D DE      253 203 D 222
SET 3, A          CBDF           203 223
SET 3, B          CBD8           203 216
SET 3, C          CBD9           203 217
SET 3, D          CBDA           203 218
SET 3, E          CBDB           203 219
SET 3, H          CBDC           203 220
SET 3, L          CBDD           203 221

SET 4, (HL)       CBE6           203 230
SET 4, (IX+D)     DDCB D E6      221 203 D 230
SET 4, (IY+D)     FDCB D E6      253 203 D 230
SET 4, A          CBE7           203 231
SET 4, B          CBE0           203 224
SET 4, C          CBE1           203 225
SET 4, D          CBE2           203 226
SET 4, E          CBE3           203 227
SET 4, H          CBE4           203 228
```

```
SET 4, L          CBE5          203 229

SET 5, (HL)       CBEE          203 238
SET 5, (IX+D)     DDCB D EE     221 203 D 238
SET 5, (IY+D)     FDCB D EE     253 203 D 238
SET 5, A          CBEF          203 239
SET 5, B          CBE8          203 232
SET 5, C          CBE9          203 233
SET 5, D          CBEA          203 234
SET 5, E          CBEB          203 235
SET 5, H          CBEC          203 236
SET 5, L          CBED          203 237

SET 6, (HL)       CBF6          203 246
SET 6, (IX+D)     DDCB D F6     221 203 D 246
SET 6, (IY+D)     FDCB D F6     253 203 D 246
SET 6, A          CBF7          203 247
SET 6, B          CBF0          203 240
SET 6, C          CBF1          203 241
SET 6, D          CBF2          203 242
SET 6, E          CBF3          203 243
SET 6, H          CBF4          203 244
SET 6, L          CBF5          203 245

SET 7, (HL)       CBFE          203 254
SET 7, (IX+D)     DDCB D FE     221 203 D 254
SET 7, (IY+D)     FDCB D FE     253 203 D 254
SET 7, A          CBFF          203 255
SET 7, B          CBF8          203 248
SET 7, C          CBF9          203 249
SET 7, D          CBFA          203 250
SET 7, E          CBFB          203 251
SET 7, H          CBFC          203 252
SET 7, L          CBFD          203 253

SLA (HL)          CB26          203 38
SLA (IX+D)        DDCB D 26     221 203 D 38
SLA (IY+D)        FDCB D 26     253 203 D 38
SLA A             CB27          203 39
SLA B             CB20          203 32
SLA C             CB21          203 33
```

SLA D	CB22	203 34
SLA E	CB23	203 35
SLA H	CB24	203 36
SLA L	CB25	203 37
SRA (HL)	CB2E	203 46
SRA (IX+D)	DDCB D 2E	221 203 D 46
SRA (IY+D)	FDCB D 2E	253 203 D 46
SRA A	CB2F	203 47
SRA B	CB28	203 40
SRA C	CB29	203 41
SRA D	CB2A	203 42
SRA E	CB2B	203 43
SRA H	CB2C	203 44
SRA L	CB2D	203 45
SRL (HL)	CB3E	203 62
SRL (IX+D)	DDCB D 3E	221 203 D 62
SRL (IY+D)	FDCB D 3E	253 203 D 62
SRL A	CB3F	203 63
SRL B	CB38	203 56
SRL C	CB39	203 57
SRL D	CB3A	203 58
SRL E	CB3B	203 59
SRL H	CB3C	203 60
SRL L	CB3D	203 61
SUB (HL)	96	150
SUB (IX+D)	DD96 D	221 150 D
SUB (IY+D)	FD96 D	253 150 D
SUB A	97	151
SUB B	90	144
SUB C	91	145
SUB D	92	146
SUB E	93	147
SUB H	94	148
SUB L	95	149
XOR (HL)	AE	174
XOR (IX+D)	DDAE D	221 174 D
XOR (IY+D)	FDAE D	253 174 D

XOR A	AF	175
XOR B	A8	168
XOR C	A9	169
XOR D	AA	170
XOR E	AB	171
XOR H	AC	172
XOR L	AD	173
XOR XX	EE XX	238 XX

APÊNDICE C

TABELA DE INSTRUÇÕES DO Z8Ø ORDENADAS POR CÓDIGOS HEXADECIMAIS

HEXADEC.	MNEMÔNICA	HEXADEC.	MNEMÔNICA
ØØ	NOP	CB 5Ø	BIT 2, B
Ø1 XXXX	LD BC, XXXX	CB 51	BIT 2, C
Ø2	LD (BC), A	CB 52	BIT 2, D
Ø3	INC BC	CB 53	BIT 2, E
Ø4	INC B	CB 54	BIT 2, H
Ø5	DEC B	CB 55	BIT 2, L
Ø6 XX	LD B, XX	CB 56	BIT 2, (HL)
Ø7	RLCA	CB 57	BIT 2, A
Ø8	EX AF, AF'	CB 58	BIT 3, B
Ø9	ADD HL, BC	CB 59	BIT 3, C
ØA	LD A, (BC)	CB 5A	BIT 3, D
ØB	DEC BC	CB 5B	BIT 3, E
ØC	INC C	CB 5C	BIT 3, H
ØD	DEC C	CB 5D	BIT 3, L
ØE XX	LD C, XX	CB 5E	BIT 3, (HL)
ØF	RRCA	CB 5F	BIT 3, A
1Ø XX	DJNZ, XX	CB 6Ø	BIT 4, B
11 XXXX	LD DE, XXXX	CB 61	BIT 4, C
12	LD (DE), A	CB 62	BIT 4, D
13	INC DEC DE	CB 63	BIT 4, E
14	INC D	CB 64	BIT 4, H
15	DEC D	CB 65	BIT 4, L
16 XX	LD D, XX	CB 66	BIT 4, (HL)
17	RLA	CB 67	BIT 4, A
18 XX	JR XX	CB 68	BIT 5, B
19	ADD HL, DE	CB 69	BIT 5, C

1A	LD A, (DE)	CB 6A	BIT 5, D
1B	DEC DE	CB 6B	BIT 5, E
1C	INC E	CB 6C	BIT 5, H
1D	DEC E	CB 6D	BIT 5, L
1E XX	LD E, XX	CB 6E	BIT 5, (HL)
1F	RRA	CB 6F	BIT 5, A
2Ø XX	JR NZ, XX	CB 7Ø	BIT 6, B
21 XXXX	LD HL, XXXX	CB 71	BIT 6, C
22 XXXX	LD (XXXX), HL	CB 72	BIT 6, D
23	INC HL	CB 73	BIT 6, E
24	INC H	CB 74	BIT 6, H
25	DEC H	CB 75	BIT 6, L
26 XX	LD H, XX	CB 76	BIT 6, (HL)
27	DAA	CB 77	BIT 6, A
28 XX	JR Z, XX	CB 78	BIT 7, B
29	ADD HL, HL	CB 79	BIT 7, C
2A XXXX	LD HL, (XXXX)	CB 7A	BIT 7, D
2B	DE CHL	CB 7B	BIT 7, E
2C	INC L	CB 7C	BIT 7, H
2D	DEC L	CB 7D	BIT 7, L
2E XX	LD L, XX	CB 7E	BIT 7, (HL)
2F	CPL	CB 7F	BIT 7, A
3Ø XX	JR NC, XX	CB 8Ø	RES Ø, B
31 XXXX	LD SP, XXXX	CB 81	RES Ø, C
32 XXXX	LD (XXXX), A	CB 82	RES Ø, D
33	INC SP	CB 83	RES Ø, E
34	INC (HL)	CB 84	RES Ø, H
35	DEC (HL)	CB 85	RES Ø, L
36 XX	LD (HL), XX	CB 86	RES Ø, (HL)
37	SCF	CB 87	RES Ø, A
38 XX	JR C, XX	CB 88	RES 1, B
39	ADD HL, SP	CB 89	RES 1, C
3A XXXX	LD A, (XXXX)	CB 8A	RES 1, D
3B	DEC SP	CB 8B	RES 1, E
3C	INC A	CB 8C	RES 1, H
3D	DEC A		
3E XX	LD A, XX	CB 8D	RES 1, L
3F	CCF	CB 8E	RES 1, (HL)
4Ø	LD B, B	CB 8F	RES 1, A
41	LD B, C	CB 9Ø	RES 2, B
42	LD B, D	CB 91	RES 2, C

43	LD B, E	CB 92	RES 2, D
44	LD B, H	CB 93	RES 2, E
45	LD B, L	CB 94	RES 2, H
46	LD B, (HL)	CB 95	RES 2, L
47	LD B, A	CB 96	RES 2, (HL)
48	LD C, B	CB 97	RES 2, A
49	LD C, C	CB 98	RES 3, B
4A	LD C, D	CB 99	RES 3, C
4B	LD C, E	CB 9A	RES 3, D
4C	LD C, H	CB 9B	RES 3, E
4D	LD C, L	CB 9C	RES 3, H
4E	LD C, (HL)	CB 9D	RES 3, L
4F	LD C, A	CB 9E	RES 3, (HL)
5Ø	LD D, B	CB 9F	RES 3, A
51	LD D, C	CB AØ	RES 4, B
52	LD D, D	CB A1	RES 4, C
53	LD D, E	CB A2	RES 4, D
54	LD D, H	CB A3	RES 4, E
55	LD D, L	CB A4	RES 4, H
56	LD D, (HL)	CB A5	RES 4, L
57	LD D, A	CB A6	RES 4, (HL)
58	LD E, B	CB A7	RES 4, A
59	LD E, C	CB A8	RES 5, B
5A	LD E, D	CB A9	RES 5, C
5B	LD E, E	CB AA	RES 5, D
5C	LD E, H	CB AB	RES 5, E
5D	LD E, L	CB AC	RES 5, H
5E	LD E, (HL)	CB AD	RES 5, L
5F	LD E, A	CB AE	RES 5, (HL)
6Ø	LD H, B	CB AF	RES 5, A
61	LD H, C	CB BØ	RES 6, B
62	LD H, D	CB B1	RES 6, C
63	LD H, E	CB B2	RES 6, D
64	LD H, H	CB B3	RES 6, E
65	LD H, L	CB B4	RES 6, H
66	LD H, (HL)	CB B5	RES 6, L
67	LD H, A	CB B6	RES 6, (HL)
68	LD L, B	CB B7	RES 6, A
69	LD L, C	CB B8	RES 7, B
6A	LD L, D	CB B9	RES 7, C
6B	LD L, E	CB BA	RES 7, D

6C	LD L, H	CB BB	RES 7, E
6D	LD L, L	CB BC	RES 7, H
6E	LD L, (HL)	CB BD	RES 7, L
6F	LD L, A	CB BE	RES 7, (HL)
7Ø	LD (HL), B	CB BF	RES 7, A
71	LD (HL), C	CB CØ	SET Ø, B
72	LD (HL), D	CB C1	SET Ø, C
73	LD (HL), E	CB C2	SET Ø, D
74	LD (HL), H	CB C3	SET Ø, E
75	LD (HL), L	CB C4	SET Ø, H
76	HALT	CB C5	SET Ø, L
77	LD (HL), A	CB C6	SET Ø, (HL)
78	LD A, B	CB C7	SET Ø, A
79	LD A, C	CB C8	SET 1, B
7A	LD A, D	CB C9	SET 1, C
7B	LD A, E	CB CA	SET 1, D
7C	LD A, H	CB CB	SET 1, E
7D	LD A, L	CB CC	SET 1, H
7E	LD A, (HL)	CB CD	SET 1, L
7F	LD A, A	CB CE	SET 1, (HL)
8Ø	ADD A, B	CB CF	SET 1, A
81	ADD A, C	CB DØ	SET 2, B
82	ADD A, D	CB D1	SET 2, C
83	ADD A, E	CB D2	SET 2, D
84	ADD A, H	CB D3	SET 2, E
85	ADD A, L	CB D4	SET 2, H
86	ADD A, (HL)	CB D5	SET 2, L
87	ADD A, A	CB D6	SET 2, (HL)
88	ADC A, B	CB D7	SET 2, A
89	ADC A, C	CB D8	SET 3, B
8A	ADC A, D	CB D9	SET 3, C
8B	ADC A, E	CB DA	SET 3, D
8C	ADC A, H	CB DB	SET 3, E
8D	ADC A, L	CB DC	SET 3, H
8E	ADC A, (HL)	CB DD	SET 3, L
8F	ADC A, A	CB DE	SET 3, (HL)
9Ø	SUB B	CB DF	SET 3, A
91	SUB C	CB EØ	SET 4, B

92	SUB D	CB E1	SET 4, C
93	SUB E	CB E2	SET 4, D
94	SUB H	CB E3	SET 4, E
95	SUB L	CB E4	SET 4, H
96	SUB (HL)	CB E5	SET 4, L
97	SUB A	CB E6	SET 4, (HL)
98	SBC A, B	CB E7	SET 4, A
99	SBC A, C	CB E8	SET 5, B
9A	SBC A, D	CB E9	SET 5, C
9B	SBC A, E	CB EA	SET 5, D
9C	SBC A, H	CB EB	SET 5, E
9D	SBC A, L	CB EC	SET 5, H
9E	SBC A, (HL)	CB ED	SET 5, L
9F	SBC A, A	CB EE	SET 5, (HL)
AØ	AND B	CB EF	SET 5, A
A1	AND C	CB FØ	SET 6, B
A2	AND D	CB F1	SET 6, C
A3	AND E	CB F2	SET 6, D
A4	AND H	CB F3	SET 6, E
A5	AND L	CB F4	SET 6, H
A6	AND (HL)	CB F5	SET 6, L
A7	AND A	CB F6	SET 6, (HL)
A8	XOR B	CB F7	SET 6, A
A9	XOR C	CB F8	SET 7, B
AA	XOR D	CB F9	SET 7, C
AB	XOR E	CB FA	SET 7, D
AC	XOR H	CB FB	SET 7, E
AD	XOR L	CB FC	SET 7, H
AE	XOR (HL)	CB FD	SET 7, L
AF	XOR A	CB FE	SET 7, (HL)
BØ	OR B	CB FF	SET 7, A
B1	OR C	DD Ø9	ADD IX, BC
B2	OR D	DD 19	ADD IX, DE
B3	OR E	DD 21 XXXX	LD IX, XXXX
B4	OR H	DD 22 XXXX	LD (XXXX), IX
B5	OR L	DD 23	INC IX
B6	OR (HL)	DD 29	ADD IX, IX
B7	OR A	DD 2A XXXX	LD IX, (XXXX)
B8	CP B	DD 2B	DEC IX
B9	CP C	DD 34 XX	INC (IX+D)
BA	CP D	DD 35 XX	DEC (IX+D)

```
BB          CP E
BC          CP H
BD          CP L
BE          CP (HL)
BF          CP A
CØ          RET NZ
C1          POP BC
C2 XXXX     JP NZ, XXXX
C3 XXXX     JP XXXX
C4 XXXX     CALL NZ, XXXX
C5          PUSH BC
C6 XX       ADD A, XX
C7          RST Ø
C8          RET Z
C9          RET
CA XXXX     JP Z, XXXX
CC XXXX     CALL Z, XXXX
CD XXXX     CALL XXXX
CE XX       ADC A, XX
CF          RST 8
DØ          RET NC
D1          POP DE
D2 XXXX     JP NC, XXXX
D3 XX       OUT (XX), A
D4 XXXX     CALL NC, XXXX
D5          PUSH DE
D6 XX       SUB XX
D7          RST 1Ø
D8          RET C
D9          EXX
DA XXXX     JP C, XXXX
DB XX       IN A, (XX)
DC XXXX     CALL C, XXXX
DE XX       SBC A, XX
DF          RST 18
EØ          RET PO
E1          POP HL
E2 XXXX     JP PO, XXXX
E3          EX (SP), HL
E4 XXXX     CALL PO, XXXX
E5          PUSH HL
```

```
DD 36 XX 2Ø LD (IX+D), XX
DD 39       ADD IX, SP
DD 46 XX    LD B, (IX+D)
DD 4E XX    LD C, (IX+D)
DD 56 XX    LD D, (IX+D)
DD 5E XX    LD E, (IX+D)
DD 66 XX    LD H, (IX+D)
DD 6E XX    LD L, (IX+D)
DD 7Ø XX    LD (IX+D), B
DD 71 XX    LD (IX+D), C
DD 72 XX    LD (IX+D), D
DD 73 XX    LD (IX+D), E
DD 74 XX    LD (IX+D), H
DD 75 XX    LD (IX+D), L
DD 77 XX    LD (IX+D), A
DD 7E XX    LD A, (IX+D)
DD 86 XX    ADD A, (IX+D)
DD 8E XX    ADC A, (IX+D)
DD 96 XX    SUB (IX+D)
DD 9E XX    SBC A, (IX+D)
DD A6 XX    AND (IX+D)
DD AE XX    XOR (IX+D)
DD B6 XX    OR (IX+D)
DD BE XX    CP (IX+D)
DD E1       POP IX
DD E3       EX (SP), IX
DD E5       PUSH IX
DD E9       JP (IX)
DD F9       LD SP, IX
DD CB XX Ø6 RLC (IX+D)
DD CB XX ØE RRC (IX+D)
DD CB XX 16 RL (IX+D)
DD CB XX 1E RR (IX+D)
DD CB XX 26 SLA (IX+D)
DD CB XX 2E SRA (IX+D)
DD CB XX 3E SRL (IX+D)
DD CB XX 46 BIT Ø, (IX+D)
DD CB XX 4E BIT 1, (IX+D)
DD CB XX 56 BIT 2, (IX+D)
DD CB XX 5E BIT 3, (IX+D)
DD CB XX 66 BIT 4, (IX+D)
```

```
E6 XX       AND XX
E7          RST 2Ø
E8          RET PE
E9          JP (HL)
EA XXXX     JP PE, XXXX
EB          EX DE, HL
EC XXXX     CALL PE, XXXX
EE XX       XOR XX
EF          RST 28
FØ          RET P
F1          POP AF
F2 XXXX     JR P, XXXX
F3          DI
F4 XXXX     CALL P, XXXX
F5          PUSH AF
F6 2Ø XX    OR XX
F7          RST 3Ø
F8          RET M
F9          LD SP, HL
FA XXXX     JP M, XXXX
FB          EI
FC XXXX     CALL M, XXXX
FE 2Ø XX    CP XX
FF          RST 38
CB ØØ       RLC B
CB Ø1       RLC C
CB Ø2       RLC D
CB Ø3       RLC E
CB Ø4       RLC H
CB Ø5       RLC L
CB Ø6       RLC (HL)
CB Ø7       RLC A
CB Ø8       RRC B
CB Ø9       RRC C
CB ØA       RRC D
CB ØB       RRC E
CB ØC       RRC H
CB ØD       RRC L
CB ØE       RRC (HL)
CB ØF       RRC A
CB 1Ø       RL B
```

```
DD CB XX 6E BIT 5, (IX+D)
DD CB XX 76 BIT 6, (IX+D)
DD CB XX 7E BIT 7, (IX+D)
DD CB XX 86 RES Ø, (IX+D)
DD CB XX 8E RES 1, (IX+D)
DD CB XX 96 RES 2, (IX+D)
DD CB XX 9E RES 3, (IX+D)
DD CB XX A6 RES 4, (IX+D)
DD CB XX AE RES 5, (IX+D)
DD CB XX B6 RES 6, (IX+D)
DD CB XX BE RES 7, (IX+DØ
DD CB XX C6 SET Ø, (IX+D)
DD CB XX CE SET 1, (IX+D)
DD CB XX D6 SET 2, (IX+D)
DD CB XX DE SET 3, (IX+D)
DD CB XX E6 SET 4, (IX+D)
DD CB XX EE SET 5, (IX+D)
DD CB XX F6 SET 6, (IX+D)
DD CB XX FE SET 7, (IX+D)
ED 4Ø             IN B, (C)
ED 41             OUT (C), B
ED 42             SBC HL, BC
ED 43 XXXX        LD (XXXX), BC
ED 44             NEG
ED 45             RET N
ED 46             IM Ø
ED 47             LD I, A
ED 48             IN C, (C)
ED 49             OUT (C), C
ED 4A             ADC HL, BC
ED 4B XXXX        LD BC, (XXXX)
ED 4D             RET I
ED 5Ø             IN D, (C)
ED 51             OUT (C), D
ED 52             SBC HL, DE
ED 53 XXXX        LD (XXXX), DE
ED 56             IM 1
ED 57             LD A, I
ED 58             IN E, (C)
ED 59             OUT (C), E
ED 5A             ADC HL, DE
```

Código	Instrução
CB 11	RL C
CB 12	RL D
CB 13	RL E
CB 14	RL H
CB 15	RL L
CB 16	RL (HL)
CB 17	RL A
CB 18	RR B
CB 19	RR C
CB 1A	RR D
CB 1B	RR E
CB 1C	RR H
CB 1D	RR L
CB 1E	RR (HL)
CB 1F	RR A
CB 2Ø	SLA B
CB 21	SLA C
CB 22	SLA D
CB 23	SLA E
CB 24	SLA H
CB 25	SLA L
CB 26	SLA (HL)
CB 27	SLA A
CB 28	SRA B
CB 29	SRA C
CB 2A	SRA D
CB 2B	SRA E
CB 2C	SRA H
CB 2D	SRA L
CB 2E	SRA (HL)
CB 2F	SRA A
CB 38	SRL B
CB 39	SRL C
CB 3A	SRL D
CB 3B	SRL E
CB 3C	SRL H
CB 3D	SRL L
CB 3E	SRL (HL)
CB 3F	SRL A
CB 4Ø	BIT Ø, B
CB 41	BIT Ø, C
ED 5B XXXX	LD DE, (XXXX)
ED 5E	IM 2
ED 6Ø	IN H, (C)
ED 61	OUT (C), H
ED 62	SBC HL, HL
ED 67	RRD
ED 68	IN L, (C)
ED 69	OUT (C), L
ED 6A	ADC HL, HL
ED 6F	RLD
ED 72	SBC HL, SP
ED 73 XXXX	LD (XXXX), SP
ED 78	IN A, (C)
ED 79	OUT (C), A
ED 7A	ADC HL, SP
ED 7B XXXX	LD SP, (XXXX)
ED AØ	LDI
ED A1	CPI
ED A2	INI
ED A3	OUTI
ED A8	LDD
ED A9	CPD
ED AA	IND
ED AB	OUTD
ED BØ	LDIR
ED B1	CPIR
ED B2	INIR
ED B3	OTIR
ED B8	LDDR
ED B9	CPDR
ED BA	INDR
ED BB	OTDR
FD Ø9	ADD IY, BC
FD 19	ADD IY, DE
FD 21 XXXX	LD IY, XXXX
FD 22 XXXX	LD (XXXX), IY
FD 23	INC IY
FD 29	ADD IY, IY
FD 2A XXXX	LD IY, (XXXX)
FD 2B	DEC IY
FD 34 XX	INC (IY+D)

```
CB 42        BIT Ø, D
CB 43        BIT Ø, E
CB 44        BIT Ø, H
CB 45        BIT Ø, L
CB 46        BIT Ø, (HL)
CB 47        BIT Ø, A
CB 48        BIT 1, B
CB 49        BIT 1, C
CB 4A        BIT 1, D
CB 4B        BIT 1, E
CB 4C        BIT 1, H
CB 4D        BIT 1, L
CB 4E        BIT 1, (HL)
CB 4F        BIT 1, A
FD 35 XX     DEC (IY+D)
FD 36 XX 2Ø  LD (IY+D), XX
FD 39        ADD IY, SP
FD 46 XX     LD B, (IY+D)
FD 4E XX     LD C, (IY+D)
FD 56 XX     LD D, (IY+D)
FD 5E XX     LD E, (IY+D)
FD 66 XX     LD H, (IY+D)
FD 6E XX     LD L, (IY+D)
FD 7Ø XX     LD (IY+D), B
FD 71 XX     LD (IY+D), C
FD 72 XX     LD (IY+D), D
FD 73 XX     LD (IY+D), E
FD 74 XX     LD (IY+D), H

FD 75 XX     LD (IY+D), L
FD 77 XX     LD (IY+D), A
FD 7E XX     LD A, (IY+D)
FD 86 XX     ADD A, (IY+D)
FD 8E XX     ADC A, (IY+D)
FD 96 XX     SUB (IY+D)
FD 9E XX     SBC (IY+D)
FD A6 XX     AND (IY+D)
FD AE XX     XOR (IY+D)
FD B6 XX     OR (IY+D)
FD BE XX     CP (IY+D)
FD E1        POP IY
FD E3        EX (SP), IY
FD E5        PUSH IY
FD E9        JP (IY)
FD F9        LD SP, IY
FD CB XX Ø6RLC (IY+D)
FD CB XX ØERRC (IY+D)
FD CB XX 16RL (IY+D)
FD CB XX 1ERR (IY+D)
FD CB XX 26SLA (IY+D)
FD CB XX 2ESRA (IY+D)
FD CB XX 3ESRL (IY+D)
FD CB XX 46BIT Ø, (IY+D)
FD CB XX 4EBIT 1, (IY+D)
FD CB XX 56BIT 2, (IY+D)
```

```
FD CB XX 5EBIT 3, (IY+D)
FD CB XX 66BIT 4, (IY+D)
FD CB XX 6EBIT 5, (IY+D)
FD CB XX 76BIT 6, (IY+D)
FD CB XX 7EBIT 7, (IY+D)
FD CB XX 86RES Ø, (IY+D)
FD CB XX 8ERES 1, (IY+D)
FD CB XX 96RES 2, (IY+D)
FD CB XX 9ERES 3, (IY+D)
FD CB XX A6RES 4, (IY+D)
FD CB XX AERES 5, (IY+D)
FD CB XX B6RES 6, (IY+D)
FD CB XX BERES 7, (IY+D)
FD CB XX C6SET Ø, (IY+D)
FD CB XX CESET 1, (IY+D)
FD CB XX D6SET 2, (IY+D)
FD CB XX DESET 3, (IY+D)
FD CB XX E6SET 4, (IY+D)
FD CB XX EESET 5, (IY+D)
FD CB XX F6SET 6, (IY+D)
FD CB XX FESET 7, (IY+D)
```

APÊNDICE D

FLAGS (Bandeiras Indicadoras de estado)

Neste apêndice pode-se observar como as instruções do Z8Ø afetam as flags, que são os bits do registro F, também conhecidas como bandeiras indicadoras de estado.

São listadas todas as instruções do Z8Ø, apresentando à frente o respectivo efeito sobre as flags. Só estão indicadas as flags importantes, que são a flag CARRY (TRANSPORTE), a flag PARITY/OVERFLOW (PARIDADE OU EXCESSO), a flag ZERO e a flag SINAL, não tendo as outras qualquer utilidade direta para o programador, uma vez que não desempenham qualquer papel na tomada de decisões (não são consideradas como condições nas instruções JR, JP, CALL e RET). Na tabela são utilizados alguns símbolos:

Nas mnemônicas:

- NN – número de um byte
- NNNN – número de dois bytes
- R – registro simples ou (HL) ou (IX+D), (IY+D) E NN
- DD – par de registros
- C – condição
- DIS – deslocamento calculado em complemento de dois

Nas flags:

- Ø – a flag é passada a zero
- 1 – a flag é passada a 1
- . – a flag não é afetada
- R – a flag é alterada em função do resultado

? – a flag assume valor aleatório

B – a flag é passada a 1 se o re – gistro B ou o par de registros BC (dependendo da instrução) contiverem zero no final da o– peração.

→ TRANSPORTE

INSTRUCÕES (MNEMÔNICAS)	SINAL	ZERO	P/O	CARRY
ADC A, R	R	R	R	R
ADC HL, DD	R	R	R	R
ADD A, R	R	R	R	R
ADD HL, DD	.	.	.	R
ADD IX, DD	.	.	.	R
ADD IY, DD	.	.	.	R
AND R	R	R	R	Ø
BIT B, R	?	R	?	.
CALL NNNN	.	.	.	.
CALL C, NNNN	.	.	.	.
CCF	.	.	.	R
CP R	R	R	R	R
CPD	R	B	R	.
CPIR	R	B	R	.
CPDR	R	B	R	.
CPL	.	.	.	.
DAA	R	R	R	R
DEC R	R	R	R	.
DEC DD	.	.	.	.
DI	.	.	.	.
DJNZ DIS	.	B	.	.
EI	.	.	.	.
EX AF, AF'	.	.	.	.
EX DE, HL	.	.	.	.
EX (SP), HL	.	.	.	.
EX (SP), IX	.	.	.	.
EX (SP), IY	.	.	.	.
EXX	.	.	.	.

	SINAL	ZERO	P/O	CARRY
HALT	.	.	.	.
IM Ø	.	.	.	.
IM 1	.	.	.	.
IM 2	.	.	.	.
INC R	R	R	R	.
INC DD	.	.	.	.
IN A , (NN)	.	.	.	.
IN R, (C)	R	R	R	.
INI	?	B	?	.
IND	?	B	?	.
INIR	?	1	?	.
INDR	?	1	?	.
JP NNNN	.	.	.	.
JP C, NNNN	.	.	.	.
JP (HL)	.	.	.	.
JP (IX)	.	.	.	.
JP (IY)	.	.	.	.
JR DIS	.	.	.	.
JR C, DIS	.	.	.	.
LD (DD), A	.	.	.	.
LD A, (DD)	.	.	.	.
LD A, R	R	R	R	.
LD A, I	R	R	R	.
LD I, A	.	.	.	.
LD R, A	.	.	.	.
LD SP, HL	.	.	.	.
LD SP, IX	.	.	.	.
LD SP, IY	.	.	.	.
LD R, R	.	.	.	.
~~LD R, NN~~	.	.	.	.
LD D, NNNN	.	.	.	.
LD A, (NNNN)	.	.	.	.
LD (NNNN), A	.	.	.	.
LD DD, (NNNN)	.	.	.	.
LD (NNNN), DD	.	.	.	.
LDI	.	.	B	.
LDD	.	.	B	.
LDIR	.	.	Ø	.
LDDR	.	.	Ø	.
NEG	R	R	R	R
NOP	.	.	.	.

	SINAL	ZERO	P/O	CARRY
OR R	R	R	R	Ø
OUT (NN), A	.	.	.	.
OUT (C), R	.	.	.	.
OUTI	?	B	?	.
OUTD	?	B	?	.
OTIR	?	1	?	.
OTDR	?	1	?	.
POP AF	R	R	R	R
POP DD	.	.	.	.
PUSH AF	.	.	.	.
PUSH DD	.	.	.	.
RES B, R	.	.	.	.
RET	.	.	.	.
RET C	.	.	.	.
RETN	.	.	.	.
RETI	.	.	.	.
RLA	.	.	.	R
RLCA	.	.	.	R
RRA	.	.	.	R
RRCA	.	.	.	R
RL R	R	R	R	R
RLC R	R	R	R	R
RR R	R	R	R	R
RRC R	R	R	R	R
RRD	R	R	R	.
RST ØØ	.	.	.	.
RST Ø8	.	.	.	.
RST 1Ø	.	.	.	.
RST 18	.	.	.	.
RST 2Ø	.	.	.	.
RST 28	.	.	.	.
RST 3Ø	.	.	.	.
RST 38	.	.	.	.
SBC A, R	R	R	R	R
SBC HL, DD	R	R	R	R
SCF	.	.	.	1
SET B, R	.	.	.	.
SLA R	R	R	R	R
SRA R	R	R	R	R
SRL R	R	R	R	R
SUB R	R	R	R	R

	SINAL	ZERO	P/O	CARRY
XOR R	R	R	R	Ø

NOTA:

Nos casos em que se indica "R" nas mnemônicas, esta letra representa não só os registros simples, como também (HL), (IX+D), (IY+D) e dados diretos "NN", quando aplicáveis.

APÊNDICE E

As variáveis do sistema

Este apêndice contém uma listagem de todas as variáveis do sistema, seus respectivos endereços, em hexadecimal e seus comprimentos em decimal. Esta área da RAM se situa entre os endereços &HF38Ø e &HFFFF. Os endereços entre HF38Ø e HF389 contém sub-rotinas em linguagem de máquina utilizadas pelo BIOS para gerenciamento de slots.

Além disso, de uma forma reduzida, as principais variáveis contêm suas funções.

Note que a listagem está por ordem alfabética, de acordo com os nomes oficiais adotados pela Microsoft.

NOME	ENDER.	COMP.	SIGNIFICADO
ARG	F847	16	
ARYTA2	F7B5	2	
ARYTAB	F6C4	2	Endereço do topo da área de matrizes.
ASPCT1	F4ØB	2	
ASPCT2	F4ØD	2	
ASPECT	F931	2	Razão de aspecto para o último CIRCLE.
ATRBAS	F928	2	Base da tabela de atributos correntes.
ATRBYT	F3F2	1	Cor da tinta gráfica.
AUTFLG	F6AA	1	Flag para modo interno de AUTO.
AUTINC	F6AD	2	Incremento de AUTO.
AUTLIN	F6AB	2	Número de linha em

			AUTO.
BAKCLR	F3EA	1	Cor de fundo.
BASROM	FBB1	1	Ativa CONTROL/STOP: ØØ=Ativado
BDRCLR	F3EB	1	Cor da borda.
BOTTOM	FC48	2	Endereço da primeira posição da RAM reconhecida pelo Interpretador.
BDRATR	FCB2	1	
BUF	F55E	258	
BUFEND	FC18	Ø	
BUFMIN	F55D	1	
CAPST	FCAB	1	Estado do CAPS:ØØ=OFF.
CASPRV	FCB1	1	
CENCNT	F933	2	
CGPBAS	F924	2	Base da tabela corrente de caracteres
CGPNT	F91F	3	
CLIKFL	FBD9	1	
CLIKSW	F3DB	1	Clic de tela: ØØ=off.
CLINEF	F935	1	
CLMLST	F3B2	1	
CLOC	F92A	2	
CLPRIM	F38C	14	Rotina usada para ler endereço do slot primário.
CMASK	F92C	1	
CNPNTS	F936	2	
CNSDFG	F3DE	1	Display das teclas de função: ØØ=off.
CODSAV	FBCC	1	
CONLO	F66A	8	
CONSAV	F668	1	
CONTXT	F666	2	
CONTYP	F669	1	
CPCNT	F939	2	
CPCNT8	F93B	2	Número de pontos do último CIRCLE.
CPLOTF	F938	1	
CRCSUM	F93D	2	

CRTCNT	F3B1	1	Número de linhas na tela.
CS1200	F3FC	10	5. variáveis para parâmetros de 1200 bauds e outras 5 para 2400 bauds.
CSAVEA	F942	2	
CSAVEM	F944	1	
CSCLXY	F941	1	
CSRSW	FCA9	1	Flag para apresentação do cursor.
CSRX	F3DD	1	Posição X do cursor de texto.
CSRY	F3DC	1	Posição Y do cursor de texto.
CSTCNT	F93F	2	
CSTYLE	FCAA	1	Tipo do cursor: 00=Bloco ....=Sublinhado
CURLIN	F41C	2	Número da linha sendo interpretada.
CXOFF	F945	2	
CYOFF	F947	2	
DAC	F7F6	16	
DATLIN	F6A3	2	Apontador de DATA.
DATPTR	F6C8	2	Endereço do topo do armazenamento do DATA
DECCNT	F7F4	1	
DECTM2	F7F2	2	
DECTMP	F7F0	2	
DEFTBL	F6CA	26	Buffer dos tipos padrões para cada grupo de variáveis Basic.
DEVICE	FD99	1	
DIMFLG	F662	1	
DONUM	F665	1	
DORES	F664	1	
DOT	F6B5	2	Número da linha atual.
DRWANG	FCBD	1	
DRWFLG	FCBB	1	

DRWSCL	FCBC	1	
DSCTMP	F698	3	
ENDBUF	F660	1	
ENDFOR	F6A1	2	
ENDPRG	F40F	5	
ENSTOP	FBB0	1	
ERRFLG	F414	1	Número do erro.
ERRLIN	F6B3	2	Número da linha com erro .
ERRTXT	F6B7	2	
ESCCNT	FCA7	1	
EXPTBL	FCC1	4	
FBUFFR	F7C5	43	
FILNAM	F866	11	Buffer que armazena o nome digitado pelo usuário.
FILNM2	F871	11	Buffer que armazena o nome lido de um dispositivo externo.
FILTAB	F860	2	
FLBMEM	FCAE	1	
FLGINP	F6A6	1	
FNKFLG	FBCE	10	
FNKSTR	F87F	160	
FNKSWI	FBCD	1	
FORCLR	F3E9	1	Cor do primeiro plano
FRCNEW	F3F5	1	Flag para distinguir CLOAD de CLOAD?: 00=CLOAD.
FRETOP	F69B	2	Endereço da próxima locação livre na área de strings.
FSTPOS	FBCA	2	
FUNACT	F7BA	2	Número de funções em uso
GETPNT	F3FA	2	Endereço do buffer de entrada do teclado
GRPACX	FCB7	2	
GRPACY	FCB9	2	
GRPATR	F3CD	2	Tabela de atributos Modo 2

GRPCGP	F3CB	2	Padrão de caracteres Modo 2
GRPCOL	F3C9	2	Tabela de cor Modo 2
GRPHED	FCA6	1	
GRPNAM	F3C7	2	Tabela de nomes Modo 2
GRPPAT	F3CF	2	Tabela de sprites Modo 2
GXPOS	FCB3	2	Posição X do cursor gráfico
GYPOS	FCB5	2	Posição Y do cursor gráfico
HEADER	F40A	1	Parâmetros do cassete
HIGH	F408	2	Parâmetros do cassete
HIMEM	FC4A	2	Endereço mais alto da memória disponível
HOLD	F83A	8	
HOLD2	F836	8	
HOLD8	F806	48	
INSFLG	FCA8	1	Flaǵ para modo inserção
INTCNT	FCA2	2	
INTFLG	FC9B	1	
INTVAL	FCA0	2	
JIFFY	FC9E	2	
KANAMD	FCAD	1	
KANAST	FCAC	1	
KBUF	F41F	318	
KEYBUF	FBF0	40	Buffer que contém os códigos dos caracteres do teclado decodifi - cados
LFPROG	F954	1	
LINL32	F3AF	1	Comprimento da tela em modo texto 32x24 (29)
LINL40	F3AE	1	Comprimento da tela em modo texto 40x24 (37)
LINLEN	F3B0	1	Comprimento da linha
LINTTB	FBB2	24	
LINWRK	FC18	40	
LOHADR	F94B	2	
LOHCNT	F94D	2	
LOHDIR	F94A	1	
LOHMSK	F949	1	
LOW	F406	24	Parâmetros do cassete
LOWLIM	FCA4	10	
LPTPOS	F415	1	Posição da cabeça da impressora
MAXDEL	F92F	2	

MAXFIL	F85F	1	
MAXUPD	F3EC	3	
MCLFLG	F958	1	
MCLLEN	FB3B	1	
MCLPTR	FB3C	2	
MCLTAB	F956	2	
MEMSIZ	F672	2	Topo da memória
MINDEL	F92D	2	
MINUPD	F3EF	3	
MLTATR	F3D7	2	Tabela de atributos Modo 3
MLTCGP	F3D5	2	Padrão de caracteres Modo 3
MLTCOL	F3D3	2	Tabela de cor Modo 3 .
MLTNAM	F3D1	2	Tabela de nomes Modo 3
MLTPAT	F3D9	2	Tabela de sprites Modo 3
MOVCNT	F951	2	
MUSICF	FB3F	1	
NAMBAS	F922	2	Base da tabela de nomes atual
NEWKEY	FBE5	11	
NLONLY	F87C	1	
NOFUNS	F7B7	1	
NTMSXP	F417	1	Flag da impressora MSX
NULBUF	F862	2	
OLDKWY	FBDA	11	
OLDLIN	F6BE	2	Próxima linha
OLDSCR	FCBØ	1	
OLDTXT	F6CØ	2	Próxima declaração
ONEFLG	F6BB	1	Flag de ON ERROR GOTO
ONELIN	F6B9	2	Número da linha do ON ERROR GOTO
ONGSBF	FBD8	1	
OPRTYP	F664	Ø	
PADX	FC9D	1	
PADY	FC9C	1	
PARM1	F6E8	1ØØ	
PARM2	F75Ø	1ØØ	
PATBAS	F926	2	Base da tabela corrente de padrão de sprites
PATWRK	FC4Ø	8	Armazenamento de um padrão de pixel de 8x8
PDIREC	F953	1	
PLYCNT	FB4Ø	1	
PRMFLG	F7B4	1	

PRMLEN	F6E6	2	
PRMLN2	F74E	2	
PRMPRV	F74C	2	
PRMSTK	F6E4	2	
PROCNM	FD89	16	
PRSCNT	FB35	1	
PRTFLG	F416	1	Flag de saída para a impressora.
PTRFIL	F864	2	
PTRFLG	F6A9	1	
PUTPNT	F3F8	2	Endereço do buffer de saída do teclado.
QUEBAK	F971	4	
QUETAB	F959	24	24 variáveis que formam os blocos de controle para as 3 filas musicais.
QUEUEN	FB3E	1	
QUEUES	F3F3	2	
RAWPRT	F418	1	Flag para impressão.
RDPRIM	F380	5	Leitura do conector primário.
REPCNT	F3F7	1	Contador do tempo de repetição de tecla(01)
RG0SAV	F3DF	1	Estado dos 8 registros de escrita do VDP.
RG1SAV	F3E0	1	
RG2SAV	F3E1	1	
RG3SAV	F3E2	1	
RG4SAV	F3E3	1	
RG5SAV	F3E4	1	
RG6SAV	F3E5	1	
RG7SAV	F3E6	1	
RNDX	F857	8	
RS2IQ	FAF5	64	Fila da RS232.
RTPROG	F955	1	
RTYCNT	FC9A	1	
RUNBNF	FCBE	1	
RUNFLG	F866	0	
SAVEND	F87D	2	Endereço do fim do bloco do BSAVE.
SAVENT	FCBF	2	Endereço de entrada do

			BSAVE ou BLOAD.
SAVSP	FB36	2	
SAVSTK	F6B1	2	
SAVTXT	F6AF	2	
SAVCOL	FB39	2	
SCNCNT	F3F6	1	Tecla de temporização da varredura (Ø1).
SCRMOD	FCAF	1	Modo da tela: ØØ=4Øx24 Modo Ø Ø1=32x24 Modo 1 Ø2=Modo 2 - Gráfico Ø3=Modo 3 - Multicolor
SKPCNT	F94F	2	
SLTATR	FCC9	64	
SLTTBL	FCC5	4	
SLTWRK	FDØ9	128	
STATFL	F3E7	1	(CA)-Conteúdo do reg. de estado do VDP.
STKTOP	F674	2	Topo da pilha do Z8Ø.
STREND	F6C6	2	Endereço do fim do armazenamento.
SUBFLG	F6A5	1	
SWPTMP	F7BC	8	
T32ATR	F3C3	2	Tabela de atributos Modo 1
T32CGP	F3C1	2	Padrão de caracteres Modo 1
T32COL	F3BF	2	Tabela de cor Modo 1.
T32NAM	F3BD	2	Tabela de nome Modo 1.
T32PAT	F3C5	2	Tabela de sprites Modo 1
TEMP	F6A7	2	
TEMP2	F6BC	2	
TEMP3	F69D	2	
TEMP8	F69F	2	
TEMP9	F7B8	2	
TEMPPT	F678	2	
TEMPST	F67A	3Ø	Buffer para descrição de strings.
TRCFLG	F7C4	1	Flag TRON/TROFF.
TRGFLG	F3E8	1	(F1)-Estado dos botões de tiro dos joysticks ou da barra de espaço.
TRPTBL	FC4C	78	

TTYPOS	F661	1	
TXTATR	F3B9	2	Tabela de atributos Modo Ø
TXTCGP	F3B7	2	Padrão de caracteres modo Ø.
TXTCOL	F3B5	2	Tabela de cor modo Ø.
TXTNAM	F3B3	2	Tabela de nome modo Ø.
TXTPAT	F3BB	2	Tabela de sprites modo Ø.
TXTTAB	F676	2	Endereço do primeiro byte da área de programas.
USFLG	F6A6	Ø	
USRTAB	F39A	2Ø	Endereços de USRØ a USR9.
VALTYP	F663	1	Código do tipo de operando:2=Inteiro; 3=String; 4=Precisão simples; 8=Prec. dupla
VARTAB	F6C2	2	Endereço do topo da área de variáveis.
VCBA	FB41	37	Parâmetros da voz 1.
VCBB	FB66	37	Parâmetros da voz 2.
VCBC	FB8B	37	Parâmetros da voz 3.
VLZADR	F419	2	
VLZDAT	F41B	1	
VOICAQ	F975	128	Fila musical da voz 1.
VOICBQ	F9F5	128	Fila musical da voz 2.
VIOCCQ	FA75	128	Fila musical da voz 3.
VOICEN	FB38	1	
WINWID	FCA5	1	
WRPRIM	F385	7	Escrita no conector primário.

APÊNDICE F

O uso dos "HOOKS" (ganchos)

Os hooks proporcionam um modo de interceptar o interpretador Basic, ou o sistema operacional, em certos pontos, permitindo processamentos adicionais ou alternativos.

Por exemplo, considere a chamada CHPUT do BIOS. Esta rotina é utilizada para imprimir textos na tela.

A seguir uma pequena listagem "disassemblada" das primeiras instruções de CHPUT:

```
PUSH HL         salva todos os registros
PUSH DE         usados
PUSH BC
PUSH AF
CALL HFDA4      chama o hook CHPUT
```

Primeiramente, os pares de registros HL, DE, BC e AF são colocados na pilha (stack).

Normalmente, o conteúdo de &HFDA4 e os 4 bytes seguintes é a instrução RET do Z8Ø, imediatamente passados para a instrução seguinte ao CALL. Entretanto, &HFDEA4 é um endereço da RAM, que pode ser alterado.

Altere 3 bytes, começando em &HFDA4, de tal forma que o controle passe para outra sub-rotina no endereço HDØØØ, ou seja,

```
FDA4    C3 00 D0        JP &HD000
```

Altere também os códigos em &HD000 desta forma:

```
POP HL
POP AF
POP DE
POP HL
RET
```

Agora os endereços no topo da pilha são 3 a mais que os endereços de onde &HFDA4 fora chamado, isto é, o endereço de retorno de CHPUT. Entretanto, se for removido da pilha, fará com que esta retorne ao estado em que se encontrava imediatamente antes da chamada de &HFDA4.

Se os próximos 4 endereços utilizarem as instruções POP acima, a pilha estará da mesma forma como estava na entrada de CHPUT, executando a instrução RET, saindo da sub-rotina CHPUT. Conseqüentemente o caractere que deveria ser impresso não mais o será!

Este exemplo demonstra como o sistema operacional pode ser interceptado. Os 5 bytes, começando em &HFDA4 são conhecidos como um "HOOK". Existem muitos outros hooks similares, em muitas outras locações do sistema operacional, bem como do interpretador Basic, e, por esse motivo, estão listados neste apêndice, em ordem alfabética.

NOME	ENDER.	OBSERVACÕES
H-ATTR	FE1C	MSXSTS no início da rotina ATTR$ (atributo)
H-BAKU	FEAD	SPCDSK na rotina BAKUPT, de BACK-UP
H-BINL	FE76	SPCDSK na rotina BINLOD
H-BINS	FE71	SPCDSK na rotina BINSAV

```
H-BUFL  FF8E   BINTRP na rotina BUFLIN
H-CHGE  FDC2   MSXIO no início da rotina
               CHGET (CHARACTER GET)
H-CHPU  FDA4   MSXIO no início da rotina
               CHPUT (CHARACTER PUT)
H-CHRG  FF48   BINTRP
H-CLEA  FEDØ   BIMISC, na rotina CLEARC
H-CMD   FEØD   MSXSTS no início da rotina
               CMD (COMANDO)
H-COMP  FF57   BINTRP
H-COPY  FEØ8   MSXSTS no início da rotina
               COPY (COPIAR ARQUIVOS)
H-CRDO  FEE9   BIO da rotina CRDO
H-CRUN  FF2Ø   BINTRP
H-CRUS  FF25   BINTRP
H-CVD   FE49   MSXSTS no início da rotina
               CVD (CONVERTER DBL)
H-CVI   FE3F   MSXSTS na rotina CVI
H-CVS   FE44   MSXSTS na rotina CVS
H-DEVN  FEC1   SPCDEV na rotina DEVNAM
H-DGET  FE8Ø   SPCDSK na rotina DGET
H-DIRD  FF11   BINTRP na entrada de DIRDO
H-DOGR  FEF3   GENGRP na rotina DOGRPH
H-DSKC  FEEE   BIO na rotina DSKCHI
H-DSKF  FE12   MSXSTS no início da rotina
               DSKF (DISCO LIVRE)
H-DSKI  FE17   MSXSTS no início da rotina
               DSKI (DISCO INPUT)
H-DSKO  FDEF   MSXSTS no início da rotina
               DSKO$ (DISCO OUTPUT)
H-DSPC  FDA9   MSXIO no início da rotina
               DSPCSR (CURSOR DO DISPLAY)
H-DSPF  FDB3   MSXIO na rotina DSPFNK
H-EOF   FEA3   SPCDSK na função EOF
H-ERAC  FDAE   MSXIO na rotina ERACSR
H-ERAF  FDB8   MSXIO na rotina ERAFNK
H-ERRF  FFØ2   BINTRP
H-ERRO  FFB1   BINTRP na rotina ERROR
H-ERRP  FEFD   BINTRP na rotina ERRPRT
H-EVAL  FF7Ø   BINTRP
H-FIEL  FE2B   MSXSTS na rotina FIELD
```

```
H-FILE   FE7B    SPCDSK no comando FILES
H-FILO   FE85    SPCDSK na rotina FILOU1
H-FINE   FF1B    BINTRP
H-FING   FF7A    BINTRP
H-FINI   FF16    BINTRP
H-FINP   FF5C    BINTRP
H-FORM   FFAC    MSXIO  na rotina FORMAT
H-FPOS   FEA8    SPCDSK na função FPOS
H-FRET   FF9D    BISTRG na rotina FRETM
H-FRME   FF66    BINTRP
H-FRQI   FF93    BINTRP na rotina FRQINT
H-GEND   FEC6    SPCDEV na rotina GENDSP
H-GETP   FE4E    SPCDSK na rotina GETPTR
H-GONE   FF43    BINTRP
H-INDS   FE8A    SPCDSK da rotina INDSKC
H-INIP   FDC7    MSXIO no início da rotina
                 INIPAT
H-INLI   FDE5    MSXINL no início da rotina
                 LININ
H-IPL    FE03    MSXSTS no início da rotina
                 IPL (INITIAL PROGRAM LOAD)
H-ISFL   FEDF    BIMISC na rotina ISFLIO
H-ISMI   FF7F    BINTRP na rotina ISMID$
H-ISRE   FF2A    BINTRP
H-KEYC   FDCC    MSXIO no início da rotina
                 KEYCOD
H-KEYI   FD9A    MSXIO no início da manipu -
                 lação de interrupções
H-KILL   FDFE    MSXSTS no início da rotina
                 KILL
H-KYEA   FDD1    MSXIO no início da rotina
                 KYEASY
H-LIST   FF89    BINTRP na rotina LIST
H-LOC    FE99    SPCDSK na função LOC
H-LOF    FE9E    SPCDSK na função LOF
H-LOPD   FED5    BIMISC na rotina LOPDFT
H-LPTO   FFB6    MSXIO na rotina LPTOUT
H-LPTS   FFBB    MSXIO na rotina LPTSTT
H-LSET   FE21    MSXSTS no início da rotina
                 LSET
H-MAIN   FF0C    BINTRP na entrada principal
```

H-MERG	FE67	SPCDSK na rotina MERGE
H-MKD$	FE3A	MSXSTS no início da rotina MKD$
H-MKI$	FE30	MSXSTS no início da rotina MKI$
H-MKS$	FE35	MSXSTS no início da rotina MKS$
H-NAME	FDF9	MSXSTS no início da rotina NAME
H-NEWS	FF3E	BINTRP
H-NMI	FDD6	MSXIO no início da rotina NMI
H-NODE	FEB7	SPCDEV na rotina NODEVN
H-NOFO	FE58	SPCDSK na rotina NOFOR
H-NOTR	FF34	BINTRP
H-NTFL	FE62	SPCDSK na rotina NTFLO
H-NTFN	FF2F	BINTRP
H-NTPL	FF6B	BINTRP
H-NULO	FE5D	SPCDSK na rotina NULOPN
H-OKNO	FF75	BINTRP
H-ONGO	FDEA	MSXSTS na rotina ONGOTP
H-OUTD	FEE4	BIO da rotina OUTDO
H-PARD	FEB2	SPCDEV na rotina PARDEV
H-PKYD	FFA7	MSXIO na rotina PHYDIO
H-PINL	FDDB	MSXINL no início da rotina PINLIN (PROGRAM LINE)
H-PLAY	FFC5	MSXSTS na entrada da declaração PLAY
H-POSD	FEBC	SPCDEV na rotina POSDSK
H-PRGE	FEF8	BINTRP na rotina PRGEND
H-PRTF	FF52	BINTRP
H-PTRG	FFA2	BIPTRG na rotina PTRGET
H-QINL	FDE0	MSXINL no início da rotina QINLIN
H-READ	FF07	BINTRP
H-RETU	FF4D	BINTRP
H-RSET	FE26	MSXSTS no início da rotina RSET
H-RSLF	FE8F	SPCDSK para reselecionar drive anterior
H-RUNC	FECB	BIMISC na rotina RUNC

H-SAVD	FE94	SPCDSK para salvar drive corrente
H-SAVE	FE6C	SPCDSK na rotina SAVE
H-SCNE	FF98	BINTRP
H-SCRE	FFCØ	MSXSTS na entrada da declaração SCREEN
H-SETF	FE53	SPCDSK na rotina SETFIL
H-SETS	FDF4	MSXSTS no início da rotina SETS
H-SNGF	FF39	BINTRP
H-STKE	FEDA	BIMISC na rotina STKERR
H-TIMI	FD9F	MSXIO no início da manipulação das interrupções
H-TOTE	FDBD	MSXIO no início da rotina TOTEXT
H-TRMN	FF61	BINTRP
H-WIDT	FF84	BINTRP na rotina WIDTHS

APÊNDICE G

Tabela de caracteres padrão ASCII e ABICOMP

Código Decimal	Código Hexadecimal	Caractere ASCII	Caractere ABICOMP
32	20	espaço	
33	21	!	
34	22	"	
35	23		
36	24	$	
37	25	%	
38	26		
39	27	'	
40	28	(	
41	29	)	
42	2A	*	
43	2B	+	
44	2C	,	
45	2D	—	
46	2E	.	
47	2F	/	
48	30	0	
49	31	1	
50	32	2	
51	33	3	
52	34	4	
53	35	5	
54	36	6	
55	37	7	
56	38	8	

57	39	9
58	3A	:
59	3B	;
6Ø	3C	
61	3D	=
62	3E	
63	3F	?
64	4Ø	
65	41	A
66	42	B
67	43	C
68	44	D
69	45	E
7Ø	46	F
71	47	G
72	48	H
73	49	I
74	4A	J
75	4B	K
76	4C	L
77	4D	M
78	4E	N
79	4F	O
8Ø	5Ø	P
81	51	Q
82	52	R
83	53	S
84	54	T
85	55	U
86	56	V
87	57	W
88	58	X
89	59	Y
9Ø	5A	Z
91	5B	[
92	5C	
93	5D	]
94	5E	^
95	5F	_
96	6Ø	libra
97	61	a

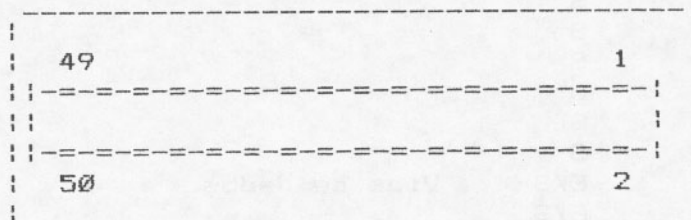
98	62	b	
99	63	c	
100	64	d	
101	65	e	
102	66	f	
103	67	g	
104	68	h	
105	69	i	
106	6A	j	
107	6B	k	
108	6C	l	
109	6D	m	
110	6E	n	
111	6F	o	
112	70	p	
113	71	q	
114	72	r	
115	73	s	
116	74	t	
117	75	u	
118	76	v	
119	78	w	
120	79	x	
121	7A	y	
122	7B	z	
123	7C		
124	7D		
125	7E		
126	7F	BS	
127	80	copyright	
160	A0		
161	A1		À
162	A2		Á
163	A3		Â
164	A4		Ã
165	A5		
166	A6		
167	A7		
168	A8		É
169	A9		Ê

170	AA	
171	AB	
172	AC	í
173	AD	
174	AE	
175	AF	
176	BØ	
177	B1	Ó
178	B2	Ô
179	B3	Õ
18Ø	B4	
181	B5	
182	B6	
183	B7	
184	B8	
185	B9	
186	BA	
187	BB	
188	BC	
189	BD	
19Ø	BE	
191	BF	
192	CØ	
193	C1	à
194	C2	á
195	C3	â
196	C4	ã
197	C5	
198	C6	ç
199	C7	
2ØØ	C8	é
2Ø1	C9	ê
2Ø2	CA	
2Ø3	CB	
2Ø4	CC	í
2Ø5	CD	
2Ø6	CE	
2Ø7	CF	
2Ø8	DØ	
2Ø9	D1	
21Ø	D2	

211	D3
212	D4
213	D5
214	D6
215	D7
216	D8
217	D9

APÊNDICE H

Pinagem do conector de cartuchos

```
 ________________________________________
|                                        |
|  49                              1     |
||-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-|   |
||                                   |   |
||-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-|   |
|  50                              2     |
|________________________________________|
```

PINO	NOME	E/S	DESCRIÇÃO
1	CS1	S	ROM H4000-H7FFF
2	CS2	S	ROM H8000-HBFFF
3	CS12	S	ROM H4000-HBFFF
4	SLTSL	S	Seleção de slot
5	-	-	
6	RFSH	S	Refrescamento
7	WAIT	E	Aguarda CPU
8	INT	E	Sinal de interrupt
9	M1	S	Ciclo da CPU
10	BUSDIR	S	Direção dos dados
11	IORQ	S	Chamada de E/S
12	MERQ	S	Chamada de memória
13	WR	S	Tempo de escrita
14	RD	S	Tempo de leitura
15	RESET	S	Reseta o sistema
16	-	-	
17	A9	S	Vias de endereços
18	A15	S	

19	A11	S	
20	A1Ø	S	
21	A7	S	
22	A6	S	
23	A12	S	
24	A8	S	
25	A14	S	
26	A13	S	
27	A1	S	
28	AØ	S	
29	A3	S	
3Ø	A2	S	
31	A5	S	
32	A4	S	
33	D1	E/S	Vias de dados
34	DØ	E/S	
35	D3	E/S	
36	D2	E/S	
37	D5	E/S	
38	D4	E/S	
39	D7	E/S	
4Ø	D6	E/S	
41	GND	–	Terra
42	CLOCK	S	Clock da CPU 3.58
43	GND	–	Terra
44	SW1	–	Detecção de cart.
45	+5V	–	+5V CC
46	SW2	–	Detecção de cart.
47	+5V	–	+5V CC
48	+12V	–	+12V CC
49	SOUNDIN	E	Entrada de som
5Ø	-12V	–	-12V CC

OUTROS LIVROS NA ÁREA

Burd / Moreira	– MSX – Guia do Operador
Burd / Moreira	– MSX – Jogos – volumes I, II e III
Burd	– Simulações no MSX
Casari	– MSX com Disk Drive
Bussab	– MSX – Música
Hoffman	– MSX – Guia do Usuário

0-07-450078-3

McGraw-Hill